MASTER EN SCIENCES INFORMATIQUES

Proposition et analyse d'un éditeur graphique

Ladam, Jean-Paul; Lefebvre, Jacques

Award date:
1984

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR (BELGIQUE),

INSTITUT D'INFORMATIQUE

PROPOSITION ET ANALYSE D'UN EDITEUR GRAPHIQUE

Jean-Paul LADAM

Jacques LEFEBVRE

Promoteur : Cl. Cherton

Mémoire présenté en vue
de l'obtention du grade de
LICENCIE ET MAITRE EN INFORMATIQUE

ANNEE ACADEMIQUE 1983 - 1984

Remarque :

Toute référence à la bibliographie, que l'on pourra retrouver dans les pages qui suivent, sera notée de la manière suivante :
2 lettres suivies de 2 chiffres, le tout entre paranthèses.

PARTIE 1 : INTRODUCTION

(Lefebvre J.)

- L'application de l'informatique au traitement graphique est de plus en plus courante et tend à s'introduire dans un nombre sans cesse croissant de domaines aussi variés que, par exemple, la gestion, l'éducation, l'art graphique.
  Elle permet d'obtenir, automatiquement ou d'une manière interactive, une représentation claire et concise d'un ensemble complexe d'informations.
  Le traitement graphique n'apparaissant pas encore dans le programme de cours de l'Institut d'Informatique, nous avons jugé intéressant de consacrer notre travail de fin d'étude à un sujet en pleine expansion et entièrement nouveau pour nous.

- Ce mémoire aborde le problème de l'édition graphique sur microordinateur.

- Dans un premier temps,nous étudierons ce domaine d'un point de vue général : nous tenterons d'établir une synthèse élémentaire et partielle, relative à l'utilisation actuelle des outils graphiques. Cette étude nous paraissait utile, vu que le problème n'avait pas été abordé lors de nos études. Cela constituera la deuxième partie.

- La troisième partie sera consacrée à une description précise de l'éditeur que nous choisissons de construire et des concepts qu'il manipule.
  L'objectif de l'utilisateur, lorsqu'il travaille avec un tel outil étant d'éditer des dessins, nous avons tenté d'aborder cette notion d'une manière particulière : un dessin sera composé d'un ensemble de dessins (notion de combinaison de dessins).
  Nous en distinguerons deux types:

  * les dessins dits de base et fournis par le système,

  * les dessins dits composés et créés par l'utilisateur à l'aide de l'éditeur, grâce à cette notion de combinaison.

  Suite à une telle définition d'un dessin, nous pourrons dire qu'ils possèderont une structure arborescente, chaque noeud de cette structure étant un dessin.
  Nous avons ensuite choisi de regrouper les dessins se ressemblant (c'est-à-dire possédant la même structure d'arbre) en famille qui

seront décrites par ce que l'on a appelé un "opérateur".
Les caractéristiques communes aux dessins se ressemblant, définissent la famille de dessins, les caractéristiques différentes définissant un élément de cette famille (c'est-à-dire un dessin particulier).

Le type de la famille dépendant des dessins qu'elle regroupe, nous aurons donc des opérateurs dits de base et des opérateurs dits composés. De là, au niveau de notre éditeur, on peut dire que créer une nouvelle famille revient à créer un nouvel opérateur en utilisant les opérateurs existants (notion de combinaison d'opérateurs). Les opérateurs ainsi utilisés constituent les noeuds du deuxième niveau de la structure du nouvel opérateur.

Il nous fallait alors concevoir un langage, langage dans lequel on écrira un opérateur composé. Nous avons défini ce que l'on a appelé le langage LG (langage graphique). En fait, un opérateur composé sera donc constitué d'un texte écrit dans ce langage. Un opérateur possédera une liste de paramètres permettant lors de l'exécution de celui-ci (avec des valeurs particulières pour chacun des paramètres), d'obtenir la représentation graphique d'un dessin particulier de la famille décrite par cet opérateur.

Au niveau construction, la création d'un nouvel opérateur était quelque chose d'assez aisé : il suffisait de combiner des opérateurs existants afin d'en obtenir de nouveaux.

Le système, tel qu'il était proposé, nous permettait d'utiliser les opérateurs existants dont on ne devait connaître que la liste des paramètres. Il aurait été intéressant de pouvoir appliquer à un opérateur que l'on utilisait dans la définition d'un autre, des opérations du style : rotation, translation, mettre en couleur ...
Nous avons choisi de fournir un outil permettant cela : "les modificateurs".
De plus, ces modificateurs pourront se rapporter à un noeud quelconque de la structure de l'opérateur utilisé.
Des règles de priorité entre modificateurs ont donc été établies.

Nous avons ensuite présenté l'éditeur graphique, éditeur qui manipulera tous ces concepts.

La description que nous avons faite de celui-ci, propose un ensemble de commandes et un ensemble de modes regroupant ceux-ci.
Grâce à cet éditeur, l'utilisateur pourra éditer le texte d'un opérateur composé et obtenir la représentation graphique d'un dessin. Bien d'autres choses sont possibles mais ces deux éléments constituent le fondement de l'éditeur.

Nous avons choisi d'approfondir particulièrement l'aspect édition d'un tel opérateur.
Editer un opérateur composé revient à éditer son texte en langage LG.
On aurait pu se contenter de fournir à l'utilisateur un éditeur de texte lui permettant de créer et/ou modifier de tels textes.
Cependant, de cette manière, l'utilisateur ne possède aucun appui graphique.
Afin de pouvoir disposer d'un tel appui, nous avons défini deux manières d'éditer un opérateur composé :

* soit créer ou modifier le texte d'un opérateur composé via un éditeur de texte. L'utilisateur éditera donc lui-même le texte en langage LG. Il ne dispose alors d'aucun appui graphique.

* soit créer ou modifier le texte d'un opérateur composé à l'aide d'un support graphique. L'utilisateur pourra créer un nouvel opérateur (famille de dessins) en créant un élément (dessin) de cette famille.
  Grâce à un support graphique, il créera un dessin à partir duquel le système générera le texte en langage LG correspondant à l'opérateur décrivant la famille d'où est issu ce dessin. En ce qui concerne la modification, l'utilisateur choisira un dessin de la famille qu'il veut modifier et il réalisera ces modifications sur la représentation graphique de celui-ci(support). Le système traduira ces modifications en changements dans le texte décrivant la famille d'où est issu ce dessin.

De plus, lors d'une même édition, l'utilisateur pourra passer d'une manière d'éditer à l'autre.
Ce double aspect de l'édition d'un opérateur composé constituera la particularité de notre éditeur.

- Nous consacrerons ensuite la quatrième partie, à analyser plus particulièrement les problèmes liés à la réalisation de deux aspects de notre éditeur :

    1. l'exécution d'un opérateur afin d'obtenir la représentation graphique d'un dessin.

    2. l'édition d'un opérateur.

  Le premier point nous amènera à aborder, entre autre, la notion de texte, et le problème des priorités entre modificateurs pour lequel nous avons proposé un algorithme en pseudo-langage.

  A propos du deuxième point, nous traiterons d'abord les deux manières d'éditer, séparément, en nous attardant plus longuement sur celle possédant un support graphique, avant d'analyser le passage de l'un à l'autre.

- Enfin, nous terminerons par une cinquième partie, constituée d'une critique et d'une évaluation du travail réalisé dans le cadre de ce mémoire.

P A R T I E 2 : C O N T E X T E

(Ladam J.-P.)

2.1. INTRODUCTION

Nous commencerons par présenter le graphique interactif de manière générale (2.2.).
Nous étudirons par la suite, l'environnement de notre sujet (2.3.).

2.2. LE GRAPHIQUE INTERACTIF (FO82)

A ce niveau, nous citerons quelques exemples relatifs au graphique interactif, ensuite nous donnerons quelques avantages d'un outil interactif et enfin, nous présenterons les différents composants d'une telle application.

1. Nous allons introduire une définition du traitement graphique en informatique par l'intermédiaire de quelques exemples familiers pour la plupart des lecteurs :

   - l'utilisation d'imprimantes à impacts permettant d'obtenir des reproductions de personnages de dessins animés ;
   - l'utilisation de tables traçantes (graphes bi ou tridimentionnels, histogrammes, cartes, dessins architecturaux, représentation de circuit) ;
   - l'utilisation d'un terminal graphique avec un clavier et un curseur, afin de manipuler des images d'objets réels ou imaginaires.

- l'utilisation d'un ordinateur individuel destiné non seulement à des usages récréatifs par l'intermédiaire d'une vaste gamme de jeux vidéo mais également à des applications classiques de traitement de l'information ;
- l'utilisation de microprocesseurs orientés exclusivement vers les jeux vidéos et reliés à une télévision.
- En C.A.O. (Conception Assistée par Ordinateur), l'utilisation du graphique interactif afin de concevoir des composants ou des systèmes, mécaniques, électriques, électro-mécaniques ou électroniques.
  ex. : coque d'un avion, un building, une usine chimique, des systèmes optiques, des réseaux téléphoniques ou informatiques.
  Sur base d'un modèle du composant ou du système, l'ordinateur permet au concepteur de réaliser des tests concernant ses propriétés mécaniques, électriques ou thermiques.

A partir de ces quelques exemples, nous pouvons remarquer que le graphique en informatique concerne la création, la mémorisation et la manipulation de modèles d'objets (pour nous dessins) et de leurs représentations graphiques par un ordinateur. En fait, on pourrait considérer le concept de dessin comme possédant un ensemble d'attributs.
Supposons par exemple : le dessin MAISON. Celui-ci possède différents attributs tels que son image, son prix, son propriétaire. Dans notre approche, nous nous intéresserons uniquement à l'attribut image.

La notion d'interactivité adjointe au graphique reprend le cas important où l'utilisateur contrôle dynamiquement l'image (représentation graphique du dessin), le format, la taille, les couleurs sur l'écran par l'intermédiaire d'outils interactifs tels qu'un clavier, un "JOYSTICK", ...

2. L'outil interactif présente certains avantages :

- une communication homme/ordinateur plus facile (traitement plus rapide et efficace de données si elles sont présentées graphiquement).
- La réalisation d'images d'objets abstraits ou réels et d'en reproduire autant de copie que l'on désire.
- Le graphique interactif est une forme d'interaction homme-machine qui combine une communication textuelle ou alphanumérique avec une communication graphique.
- L'utilisateur est libéré de la fatigue liée au parcours de nombreuses pages de texte sur listing ou sur terminal.
  La possibilité de faire varier l'image dynamiquement constitue donc un avantage appréciable.
- Pour exprimer des changements dans le temps une séquence animée est souvent plus expressive qu'une succession d'images. La vision est plus agréable.

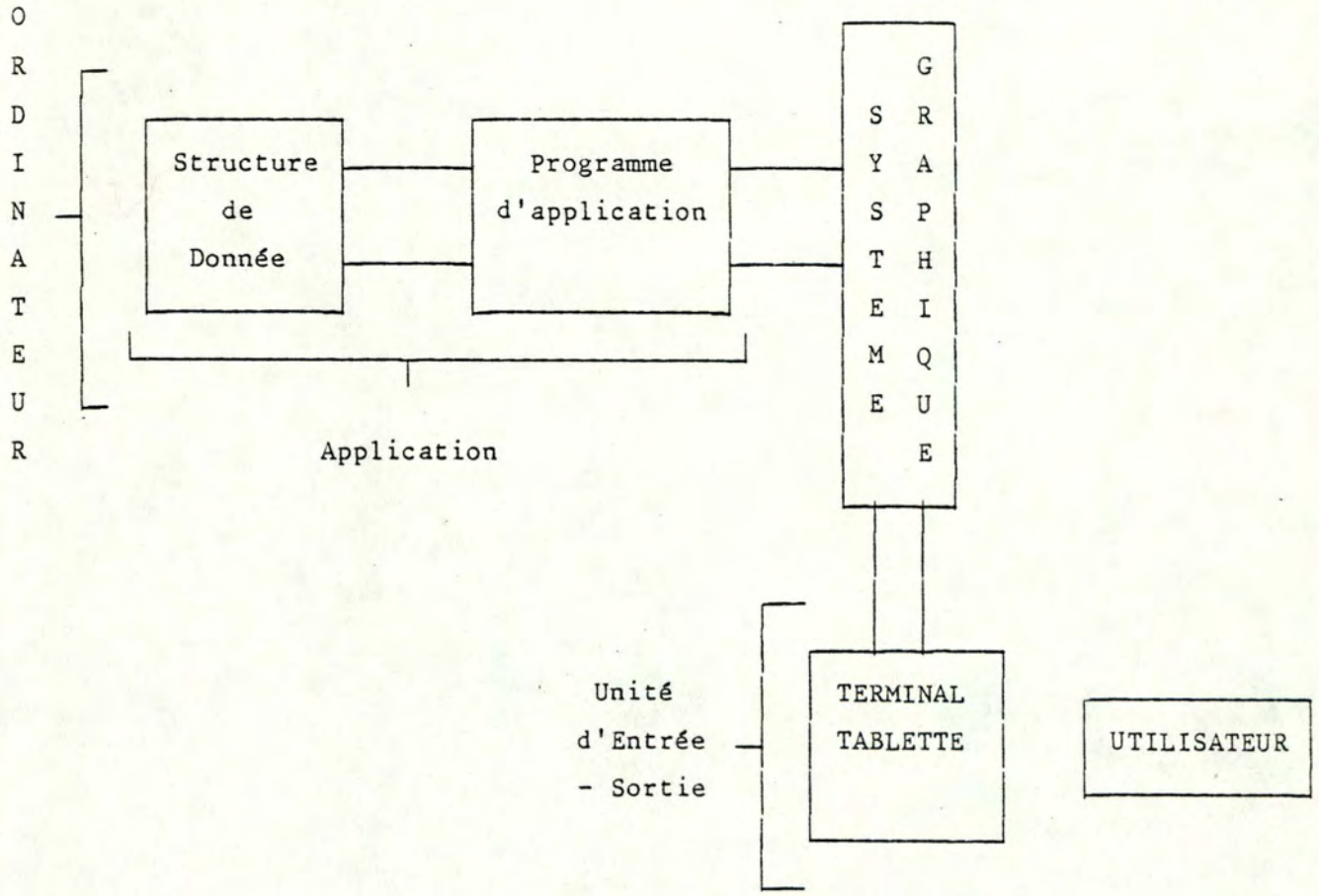
Les interactions possibles sont de différents types :

- Impression interactive où l'utilisateur fournit des paramètres, fait exprimer, modifie les paramètres, refait imprimer.
- Prédéfinir un objet et voler autour de lui en temps réel sous contrôle de l'utilisateur (simulateur de vol).
- Conception interactive : l'utilisateur part d'un écran vierge, crée un dessin à partir de composants prédéfinis, le modifie selon sa volonté, effectue des déplacements et des agrandissements.

Ce dernier point sera la préoccupation principale de notre travail.
En résumé, nous pouvons dire que le graphique interactif nous permet d'obtenir une meilleure communication homme/machine si l'on a recours à une judicieuse combinaison entre textes et images dynamiques ou statistiques.

3. Enfin, pour terminer ce point, présentons les différents composants d'une application graphique interactive.

On peut retrouver dans le schéma ci-dessous emprunté à J. Foley (F082) les différents composants de base d'une application graphique interactive :



On dispose d'un ordinateur central auquel est relié une unité d'affichage constituée d'une unité de sortie et d'une unité d'entrée.

Sortie : écran.

Entrée : clavier, touches prédéfinies, crayon lumineux, "Joystick" qui indique la position selon les axes X, Y.

Au point de vue logiciel, les composants sont au nombre de trois : le programme d'application qui enregistre et utilise le second composant : la structure de donnée de l'application et, envoie des commandes graphiques vers le 3ème composant, le système graphique.
La structure de donnée enregistre la description d'objets réels ou abstraits dont les images doivent apparaître à l'écran. (Ces informations sont : type de ligne, couleur, coordonnées). Le programme d'application décrit la géométrie de l'objet dont l'image doit être visualisée sur une surface de vue par l'intermédiaire d'un système graphique.
Le système graphique contient un ensemble de sous-routines de sorties compatibles avec un langage de haut-niveau. Ces sous-routines contrôlent une ou plusieurs unités de sortie et engendrent l'affichage de l'image sur ces unités. Il contient également un ensemble de sous-routines d'entrées d'info graphiques permettant la communication avec l'utilisateur.

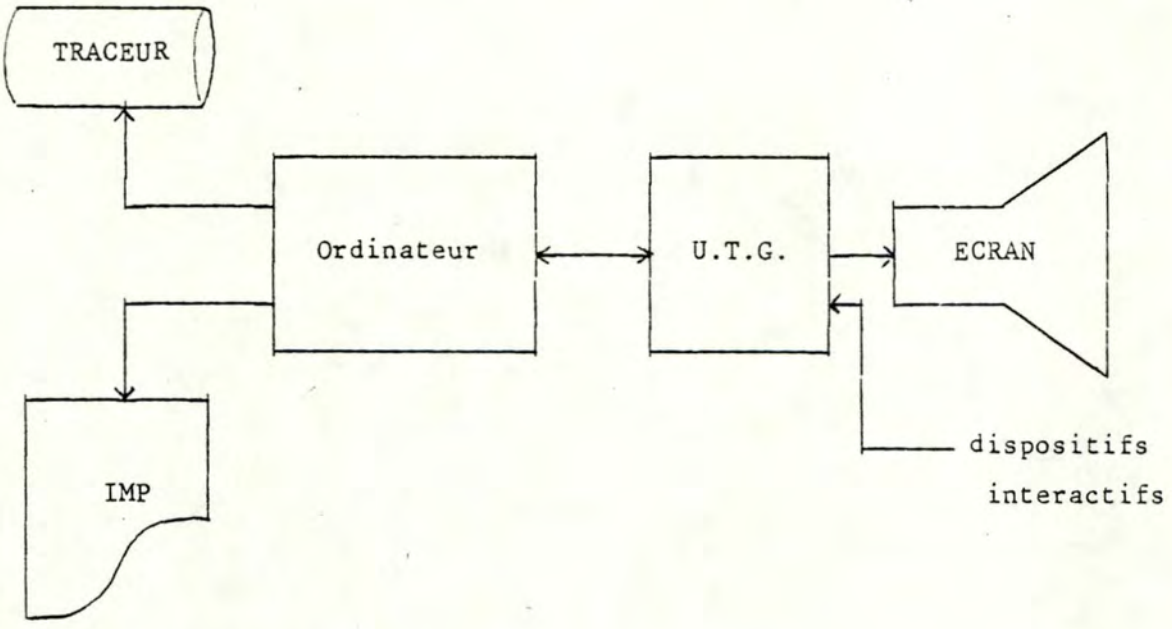
2.3. DEFINITION DE L'ENVIRONNEMENT

Après avoir étudié le graphique interactif de manière générale, nous définirons de manière plus précise les composants d'une application graphique interactive (son environnement).
Nous parlerons tout d'abord du matériel (2.3.1.), des tentatives de normalisation au sein des logiciels graphiques : les standards ainsi que le rapport de ceux-ci avec les microordinateurs (2.3.2.), ensuite nous citerons quelques applications graphiques (2.3.3.), enfin nous définirons la notion d'éditeur graphique (2.3.4.) et parlerons du langage LOGO en particulier (2.3.5.).

2.3.1. LE MATERIEL

Le matériel d'un système graphique est constitué de quatre composants principaux :

- l'ordinateur
- l'unité de traitement graphique (U.T.G.)
- l'unité d'affichage
- l'unité d'entrée donnée.



L'U.T.G. est comparable à une unité centrale spéciale avec son propre ensemble de commandes, ses propres formats de données et son propre compteur d'instructions. Il exécute une séquence d'instructions d'affichage afin de créer un dessin sur un écran. Les instructions typiques sont :

- le marquage d'un point
- le tracé d'une ligne ou d'une chaîne de caractères.

Les dispositifs interactifs sont également rattachés au U.T.G.
Nous allons citer dans ce paragraphe les différents types de terminaux graphiques, les différentes techniques d'introduction graphique de données, les différentes méthodes d'impression d'images graphiques.

2.3.1.1. TERMINAUX (FO82, MI78)

Nous ne ferons que citer les technologies principales rencontrées actuellement :

- Terminaux à tube cathodique avec rafraîchissement d'images
  2 types : . Terminaux à balayage cavalier
  . Terminaux à balayage de trame
- Terminaux à tube cathodique avec mémoire.
- Autres types de terminaux :
  . les terminaux "plasmapanel display"
  . écran à cristaux liquides (L.C.D.)
  . écran à diodes luminescentes (L.E.D.)
  . écran à lasers.

2.3.1.2. INTRODUCTION GRAPHIQUE DE DONNEES (FO82, MI78)

Il existe aujourd'hui une grande variété de techniques ; afin de les ranger, nous utiliserons la classification basée sur les dispositifs logiques définis par le Siggraph Planning Committee (S.G.P.).
Nous distinguons : "locator" : indiquer une position et une orientation (tablette + Stylo).
"pick" : sélection d'une entité à l'écran (crayon lumineux, simulé par la tablette ou la souris).
"valuator" : permet d'introduire une valeur réelle (potentiomètres rotatifs, glissants).
"keyboard" : introduire une chaîne de caractères (clavier alphanumérique).
"button" : sélection parmi un ensemble d'actions ou de choix alternatifs possibles (clavier, touche).
Choix effectué au sein d'un menu.

2.3.1.3. TECHNIQUES D'IMPRESSION DE GRAPHES

L'outil employé pour tracer des images peut être une imprimante graphique (table traçante, imprimante à rouleau, tambour, électrostatique).

2.3.1.4. NOTRE MATERIEL

Nous ne ferons que citer les différents éléments composant le matériel car nous ne réaliserons pas une implémentation complète du système. Nous supposerons donc qu'il sera mis à notre disposition :

- un micro-ordinateur (64K)
- un écran à tube cathodique (balayage de trame)
- Comme entrée de données : un clavier
- Comme sortie : une table traçante
  une imprimante.

De plus amples précisions devraient être apportées lors du choix définitif.

2.3.2. LES LOGICIELS GRAPHIQUES (FO82)

Ce paragraphe est consacré à l'étude des problèmes de standardisation des logiciels graphiques. Nous parlerons dans un premier temps du motif de la standardisation (2.3.2.1.) ensuite des exigences pour la conception d'un standard (2.3.2.2.), d'un historique (2.3.2.3.), des normes existantes à ce jour : G.S.P.C. (2.3.2.4.), G.K.S. (2.3.2.5.), N.A.P.L.P.S. (2.3.2.7.) et de la standardisation en microinformatique (2.3.2.6.).

2.3.2.1. MOTIFS

Le principal motif de la standardisation est la portabilité du programme d'application (transfert d'une installation vers une autre avec un minimum de changements dans la programmation).

Cette probabilité serait parfaite si le matériel était standard (pas envisageable). Ceci n'est réalisable que par un interface de haut niveau, c'est le logiciel graphique.
Ce logiciel sera un point de référence pour les constructeurs d'équipement graphique en fournissant une combinaison utile de capacités graphiques pour un terminal. Le problème de la dépendance vis-à-vis du matériel est illustré sur la figure ci-dessous.

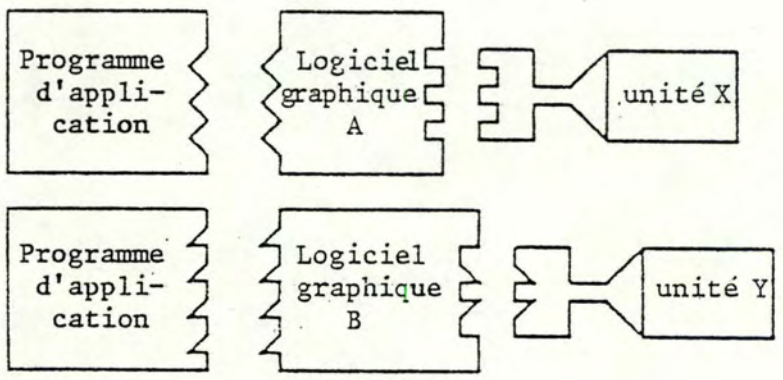


Figure a

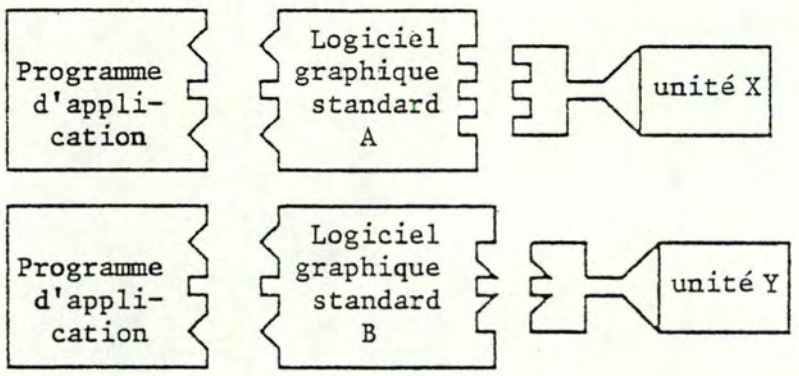


Figure b

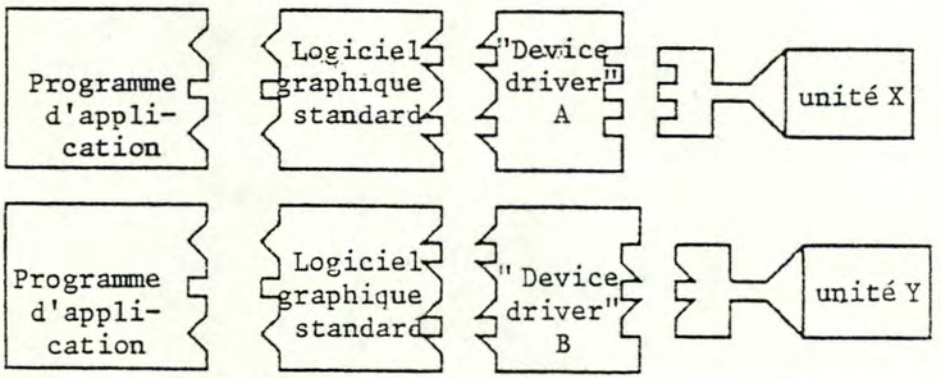


Figure c

Le premier dessin (figure a) montre deux logiciels graphiques développés pour deux terminaux différents ; chacun d'eux présentant un interface de programmation différent au programme d'application. Nous dirons que ces logiciels dépendent entièrement du terminal utilisé en ce sens qu'ils forcent le programmeur à tailler ses programmes en fonction du terminal.

Dans le deuxième dessin (figure b), nous voyons le résultat d'un effort visant à offrir un interface identique au programmeur. Le même programme d'application peut tourner sur deux sites. Nous avons donc atteint une indépendance vis-à-vis du terminal pour le programmeur ; cependant les deux logiciels graphiques diffèrent intérieurement.

Dans le troisième dessin (figure c), le logiciel graphique est indépendant du terminal. La seule partie liée au terminal est le "device driver". Sa fonction est de transformer, en fonction du terminal, le "display file", fichier contenant un ensemble d'appels à des primitives graphiques, le restant du logiciel étant tout à fait portable.

2.3.2.2. EXIGENCES POUR LA CONCEPTION D'UN STANDARD GRAPHIQUE (GK82)

La conception d'un standard graphique doit être basée sur les exigences suivantes :

- Contenir toutes les fonctions essentielles pour l'ensemble des domaines d'applications graphiques (sortie de dessins, application interactive).
- Contrôler d'une manière uniforme l'ensemble des types de terminaux graphiques.
- Fournir toutes les fonctions exigées par la majorité des applications sans devenir trop important (taille).

A partir de ces exigences, on peut établir des principes de conception, de capacité fonctionnelle, de conception d'interface, par rapport aux périphériques graphiques et l'implémentation pour concevoir les standards.

2.3.2.3. HISTORIQUE (HO82)

Par rapport aux langages de programmation, les standards graphiques ont mis longtemps pour apparaître. Le changement continuel dans la perception et la conception du graphique interactif est la raison la plus importante de ce retard.

La majorité des activités actuelles dans le domaine de la standardisation a trouvé son origine lors de la réunion organisée par l'I.F.I.P. à Seillac en mai 1976 (Seillac 1). La principale résolution de Seillac fut d'établir une distinction claire entre la modélisation d'une image et la vue que le système en donne. Suite à cette réunion le G.S.P.C. (Graphics Standard Planning Committee) conçu le "CORE graphic system". Deux propositions de standard sont apparues en 1977 et 1979. La dernière version reprend une description complète d'un système à 3 dimensions.

Pendant ce temps, un groupe allemand (D.I.N.) travaillait sur le G.K.S. (Graphical Kernel System) cherchant aussi à définir un système graphique mais limité à l'espace à 2 dimensions. La dernière version fut acceptée par l'I.S.O. (International Standard Organisation) comme proposition de base.

2.3.2.4. G.S.P.C. (MI78)

Les fonctions offertes par le CORE system sont classifiées en 4 catégories :
- Les primitives de sorties et leurs attributs (paramètres).
- Transformation de vue : Le programme d'application décrit la vue d'un objet par la spécification d'une transformation de vue.

  Une transformation de vue sélectionne une région du monde graphique à afficher et spécifie la façon dont les objets se trouvant dans la zone sélectionnée doivent être projetés sur la surface de vue.

  La surface de vue est un rectangle (ou un carré se trouvant sur la surface des périphériques utilisés.

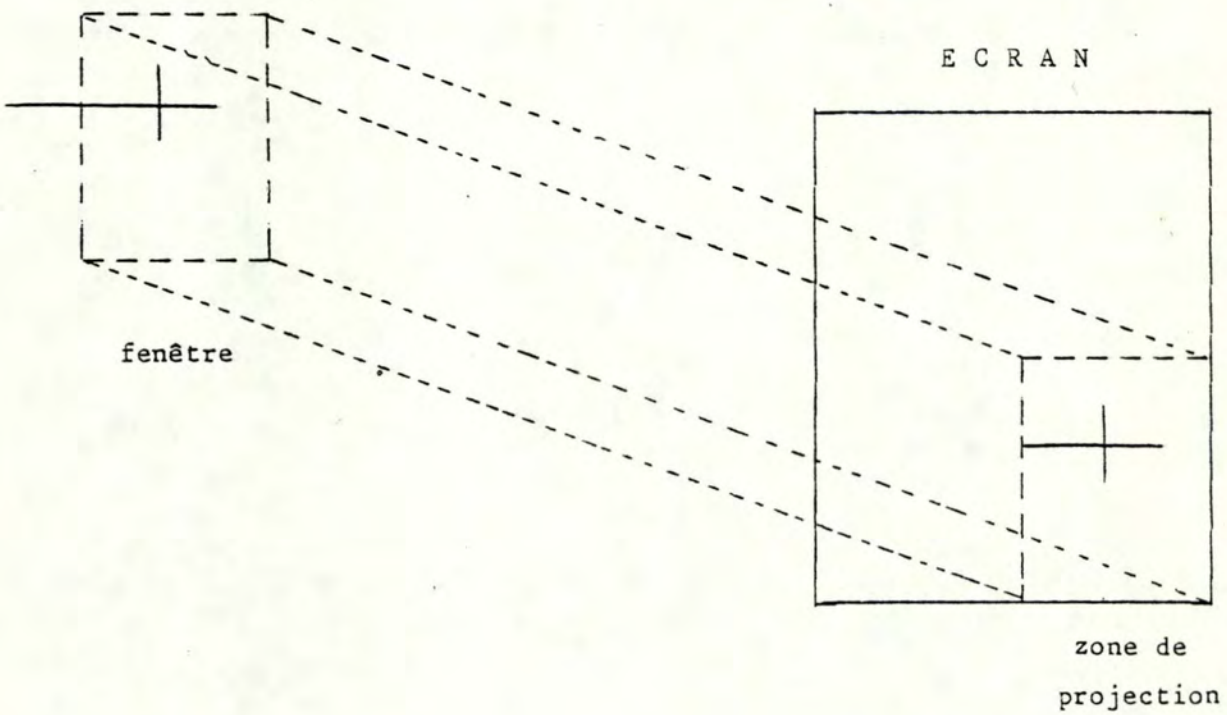
  Le système de coordonnées spécifiant les positions sur la surface de vue est appelé système de coordonnées normalisé des périphériques.

  Conceptuellement, une transformation de vue décrit la position d'une caméra qui doit prendre un instantané d'un objet.

  Dans l'espace à deux dimensions, une transformation de vue est entièrement spécifiée par une fenêtre dans le système de coordonnées utilisateur et par une zone de projection sur la surface de vue.

  La fenêtre peut être tout rectangle dans le plan déterminé par les axes X et Y du système de coordonnées utilisateur.

La zone de projection est un rectangle spécifié dans le système de coordonnées normalisé (0 ⟶ 1 pour X et 0 ⟶ 1 pour Y) dont les côtés sont parallèles aux côtés horizontaux et verticaux de la surface de vue.
Une fenêtre est utilisée pour "couper" un objet, c'est-à-dire pour déterminer la portion de l'objet devant être visualisée. Après la coupure, cette portion est projetée sur une partie de la surface de vue déterminée par la zone de projection.



- Segmentation et modification d'image : un dessin apparaissant sur la surface de vue est défini par un ou plusieurs segments. Un programme d'application utilise des segments différents afin de pouvoir changer une ou plusieurs parties d'un dessin.

Des changements dans un dessin apparaissent lorsque les attributs dynamiques des segments sont changés. (Visibilité, luminosité, détectabilité, transformation). Un segment correspond à une image. Il existe deux types de segments :

retenu : pour afficher plusieurs fois une image

non-retenu : affiche une seule fois une image.

- Dispositif logique d'entrée : Ceci permet de faire abstraction des propriétés des périphériques physiques.

Quatre classes de dispositifs : . Sélection (crayon lumineux)
. Positionnement (tablette)
. Valuator (bouton rotatif)
. Button (touche sélectionnée).

2.3.2.5. G.K.S. (GK82, HO82)

Les notions développées par G.K.S. sont au nombre de six :

- Station de travail
- Plume
- Primitives de sorties
- Segments
- Vues
- Entrées de données graphiques.

A. Station de travail

C'est le concept central qui regroupe une unité d'affichage et un certain nombre de périphériques d'entrée. Un opérateur peut avoir en même temps plusieurs stations de travail sous son contrôle. Différentes stations peuvent être utilisées pour montrer simultanément plusieurs parties d'un même dessin. La station de travail est définie comme appartenant à l'un des types standards (table traçante, écran à mémoire, écran à rafraîchissement) ce qui permet au programme d'adapter son comportement en fonction du domaine d'application.

B. Plume

Des attributs peuvent être associés à des primitives graphiques telles que couleur, type de ligne, luminosité. En G.S.P.C., les attributs gardent leur valeur courante jusqu'au changement explicite suivant. Ceci a pour désavantage de devoir adapter ces spécifications d'attributs à des périphériques ne pouvant implémenter certains attributs et un accroissement de la complexité des algorithmes pour la modification de nombreuses valeurs. En G.K.S., on possède un attribut majeur par primitive, un numéro de plume. La définition de cet attribut dépendra de la station de travail et sera établie par le programmeur d'application.

C. Primitives de sorties

G.K.S. définit six primitives de sorties :

- Polyline : permet de tracer un ensemble de lignes connectées.
- Polymarker : permet de marquer une suite de points par un symbole.
- Text : permet d'écrire une chaîne de caractères.
- Fill area : définit les limites d'une surface dont l'intérieur doit être hachuré ou colorié.
- Pixel area : moyen utilisé pour spécifier une suite de points
- Generalized drawing primitives : fonctions permettant de tracer des cercles, courbes, ...

D. Segments

Les segments sont mémorisés dans les stations de travail actives au moment de leur définition. L'utilisateur peut très bien définir un dessin formé de segments sur un terminal et désirer en avoir une copie sur une table traçante. A cette fin, G.K.S. enregistre une copie des segments. Les copies peuvent être retirées et envoyées vers n'importe quelle station. Ajoutons encore la possibilité d'insérer un segment dans un autre.

E. Vues

G.K.S. a 3 systèmes de coordonnées :

Système de coordonnées utilisateur

Système de coordonnées normalisé du périphérique

Système de coordonnées physiques du périphérique

La notion de fenêtre interviendra lors de la première projection.

F. Entrées de données graphiques

G.K.S. utilise le même système de classification que G.S.P.C.

2.3.2.6. STANDARDS ET MICROORDINATEURS (CD82)

Avec l'introduction de plusieurs centaines de systèmes graphiques différents et du grand nombre de variantes au niveau des écrans et des imprimantes, la création de langages graphiques communs et d'interfaces d'unité pour la portabilité fut nécessaire et bénéfique mais pas simple. La création d'applications imposantes, le développement d'unités d'entrées graphiques ont rendu nécessaire la création de standards. Signalons que la plupart des travaux sur les standards pour mainframe et miniordinateur sont directement utilisables pour les applications sur microordinateurs. Cependant les objectifs et contraintes des micros par rapport au mainframe et mini sont différents ; l'objectif principal des micros étant la portabilité des applications sur différents systèmes, celui des minis et mainframes étant de rendre les unités de sortie indépendantes.

Dans le monde des mainframes, la distribution du soft en code source est fréquente car peu de machines sont identiques. Tandis que pour la micro-informatique beaucoup de systèmes ont le même C.P.U. et operating system. De ce fait, les applications sont sous forme code objet. Le développement des interfaces va évoluer vers 2 possibilités ; la portabilité se portera à deux niveaux :

- portabilité du code source : un compilateur spécifique et/ou une librairie d'exécution pour les microordinateurs individuels donne au créateur d'application les outils pour créer des programmes machines graphiques spécifiques. Les modules objets de ces programmes doivent contenir un ou plusieurs "device driver" qui peuvent être choisis par des routines de sortie. Les standards peuvent définir le langage d'interfaçage pour les primitives de sortie, attribut, opération de vision, de contrôle général.
- portabilité du code objet : l'utilisation de standards rend possible l'implémentation avec un code objet indépendant. Cette approche est identique à la 1ère, à la différence que chaque créateur de compilateur n'a pas besoin d'adapter celui-ci à chaque machine.

2.3.2.7. N.A.P.L.P.S. (BE83)

Le standard graphique N.A.P.L.P.S. est orienté microprocesseur. En fait, le North American Level Protocol Syntax est une méthode d'encodage d'informations de manière standardisée et compacte. En outre N.A.P.L.P.S. est prévu pour un environnement multi-utilisateurs.
N.A.P.L.P.S. spécifie entre autre, comment formuler les dessins et les textes.
Il prévoit 2 ou 3 dimensions.

L'utilisateur de N.A.P.L.P.S. dessinera sur un écran normalisé : (0,1) * (0,1). Cependant, certains hardwares n'ont pas d'espace adressable de forme carrée : il y aura troncature du dessin. Ceci permet une indépendance du point de vue du matériel, et de plus, une simple multiplication suffit pour passer de l'écran normalisé à l'écran physique. (idem G.K.S.).
N.A.P.L.P.S. prévoit 3 modes de couleurs en fonction des applications envisagées : Mode 0 : mode élémentaire, son but est de s'adapter à tous les écrans même monochromes.
Mode 1, 2 : permet des effets chromatiques mais nécessitent un certain hardware.

N.A.P.L.P.S. spécifie les couleurs en terme de rouge, vert et bleu.
Le positionnement des caractères n'importe où sur l'écran est prévu par N.A.P.L.P.S. et la définition de nouveaux caractères est possible. Les possibilités graphiques sont celles utilisées par G.K.S. D'autres fonctions graphiques non reprises dans G.K.S. sont disponibles : ARC, ENCRLINE, ...
Notons qu'un dessin en N.A.P.L.P.S. se fait en plusieurs ensembles de deux étapes. La première étape consiste en un positionnement à l'aide d'une commande en mode absolu, la seconde consiste alors en la génération de déplacements relatifs successifs. On répète alors cette paire d'étapes pour les différentes parties du dessin. Des fonctions de contrôle sont disponibles, liées à la technique du codage.

De plus, il contient des ordres de contrôle tels que le positionnement du curseur, le clear de l'écran, ... Il est nécessaire de prévoir une zone réservée aux requêtes de l'utilisateur. N.A.P.L.P.S. spécifie que certaines surfaces de l'écran puissent être désignées comme étant réservées à l'utilisateur. Il peut rentrer des données via tous les moyens qui lui semblent adéquats tels que light pens, joysticks, tablet, ...
N.A.P.L.P.S. prévoit la définition et l'utilisation d'ensembles d'instructions : les macros. Une macro peut être envoyée de l'ordinateur vers le terminal ou inversément.

2.3.3. LES APPLICATIONS GRAPHIQUES (FO82)

Dans cette section, nous citerons simplement différentes utilisations du graphisme.

- Tracé graphique (interactif) en gestion, science et technologie. (histogrammes, tartes, courbes de fonctions économiques, mathématiques, physiques).
- Cartographie. (cartes géographiques, météorologiques, densité de population).
- Conception assistée par ordinateur. (conception de composants électroniques, de systèmes optiques, téléphoniques, informatiques).
- Simulation et animation. (Etude du comportement d'un objet dans le temps : simulateur de vol).
- Contrôle de traitement. (Ecran de contrôle dans les raffineries, contrôle nucléaire, réseaux informatiques, visualisation du trafic aérien).
- Art graphique (création de dessins par l'intermédiaire de l'outil informatique).

C'est plutôt dans cette dernière catégorie que nous nous situons.

2.3.4. NOTION D'EDITEUR GRAPHIQUE

Dans ce paragraphe nous donnerons tout d'abord une définition d'un éditeur graphique en précisant ensuite les différents composants de cette définition. Enfin nous ferons le lien entre la définition et la démarche.

2.3.4.1. DEFINITION D'UN EDITEUR GRAPHIQUE

Nous le définirons de la manière suivante :
Un éditeur graphique est un programme qui aide un utilisateur à créer et/ou transformer un dessin de manière interactive.
Notons quatre concepts clés à cette définition :

- le dessin
- l'éditeur graphique
- le dialogue
- l'utilisateur.

a) le dessin

Le dessin peut être considéré comme une suite de traits ou de points. Mais cette définition peut être étendue si les propriétés du dessin ; telles que sa structure, sont considérées. Nous reprendrons cette notion de manière plus approfondie au point 3.2.1.2.1. et 3.2.1.2.2.

b) l'éditeur graphique

L'outil informatique qui aide un utilisateur à construire un dessin est communément appelé "éditeur graphique".

c) le dialogue

Un temps de réponse très court est une des conditions pour assurer un dialogue fructueux entre l'utilisateur et l'éditeur graphique. L'interactif paraît donc adéquat pour le processus de questions-réponses. Celui-ci est possible grâce à un "interface" entre l'homme et la machine.

d) l'utilisateur

C'est un homme (ou un groupe d'hommes) qui construit un dessin. Il est clair que la notion d'être humain est générale. On peut classifier les utilisateurs en fonction de leurs objectifs. A une typologie d'utilisateurs correspond une classification des éditeurs puisque ces derniers s'adaptent aux besoins des premiers.
Dans notre cas, il s'agira d'un utilisateur tourné vers l'enseignement.

2.3.4.2. DEMARCHE SUIVIE DANS L'ANALYSE FONCTIONNELLE

Cette analyse pourrait être abordée à quatre points de vue :

- l'éditeur graphique
- l'utilisateur
- le dialogue
- le dessin.

Choisir le concept "utilisateur" comme point de départ, revient à se poser deux questions :
Quelles sont les définitions possibles d'un dessin pour l'utilisateur ?
Quelles sont, sur le dessin, les actions possibles qui intéressent l'utilisateur ?
Notre analyse sera donc influencée par ces 2 questions car l'utilisateur est le principal intéressé dans l'édition de dessin. D'ailleurs, si nous prenions le dessin ou l'éditeur graphique comme point de départ, nous ne nous poserions qu'une des deux questions citées, la première pour le dessin, la seconde pour l'éditeur graphique. Enfin la qualité du dialogue est plus une contrainte à respecter qu'un but à atteindre.

Notre démarche va donc consister à se demander ce que sont pour l'utilisateur les aspects statiques (qu'est ce qu'un dessin ?) et dynamique (quelles sont les actions sur le dessin ?) de l'éditeur graphique.

2.3.5. ETUDE PARTICULIERE D'UN LANGAGE : LOGO (AL82)

Nous consacrons un paragraphe particulier au langage LOGO, car celui-ci inspira notre démarche.

Deux aspects de LOGO nous ont intéressés : non seulement les personnes utilisant LOGO, sont orientées le plus souvent vers l'enseignement (ce qui est notre cas) mais également ce langage possède des possibilités graphiques intéressantes.
Nous nous limiterons cependant ici, à parler de ces possibilités graphiques.

LOGO se présente sous la forme d'un dictionnaire de primitives que l'utilisateur peut enrichir en définissant ses propres procédures. C'est ce que l'on appelle un langage extensible car les procédures utilisateurs sont employées de la même façon que les primitives.
Dans LOGO, on trouve 2 types de procédures : les commandes, les opérations.
Ces procédures possèdent des paramètres. Le résultat d'une opération doit toujours apparaître comme argument d'une autre procédure.
Ce langage permet de regrouper les informations sous la forme de structures de données. De plus, il est non typé (une variable peut correspondre à un nom, un nombre, une liste sans nécessiter de déclaration préalable ou d'adjonction d'un caractère spécial au nom de la variable).

Dans les procédures utilisateurs, le nombre de paramètres est toujours fixé. Cependant il peut être variable pour certaines primitives.
(Ex. : (SUM 2 3 4 5 ...)).
LOGO, à l'aide de la représentation d'une tortue se déplaçant à l'écran, permet à l'utilisateur de dessiner ce qu'il désire.
Il peut construire un dessin à l'aide de primitives de base c'est-à-dire que celles-ci lui permettent de déplacer la tortue. En fait, l'utilisateur donne successivement une suite de commandes précisant entre autre la direction, le déplacement.

Si l'utilisateur désire réutiliser ce qu'il dessine, une autre possibilité de création lui est offerte : la construction d'une commande. (Commande utilisateur).

Une commande est en fait la définition d'une famille de dessins, représentée sous la forme d'un ensemble d'actions regroupées dans un texte dans un ordre précis. Le texte est donc une suite d'instructions.

La commande peut posséder des paramètres permettant d'obtenir un dessin de la famille lorsqu'on les fixe.
De plus, elle est exécutable immédiatement après avoir été décrite.
Toute commande peut entrer dans la définition d'une autre ainsi que dans sa propre définition (récursivité).
Lorsque nous aurons défini les concepts liés à notre éditeur, nous présenterons un parallèle entre les possibilités offertes par LOGO et notre système (3.4.4.). Ainsi, nous préciserons les différents points exposés dans ce paragraphe.

P A R T I E 3 : A N A L Y S E F O N C T I O N N E L L E

(3.1. : Lefebvre J.

3.2. : Lefebvre J.

3.3. : Lefebvre J.

3.4. : Ladam J.-P.)

3.1. INTRODUCTION

Après avoir étudié le problème du graphisme et des éditeurs graphiques d'un point de vue général, nous consacrerons cette troisième partie à la description précise de l'éditeur que nous choisissons de construire et des concepts qu'il manipule.

Donc, dans un premier temps, nous allons dégager les concepts de base qui seront mis en oeuvre lors de la réalisation de notre éditeur graphique (3.2.).
Ensuite nous définirons un langage, le langage LG qui nous permettra de décrire ces concepts de base (3.3.).
Enfin nous présenterons l'éditeur graphique, éditeur qui manipulera ces concepts de base (3.4.).

3.2. CONCEPTS DE BASE

Suite à la définition d'un éditeur graphique, donnée précédemment, (un éditeur graphique est un programme qui aide l'utilisateur à créer et/ou transformer un dessin de manière interactive), il nous paraît important de d'abord voir ce que l'on entend par dessin (3.2.1.), car l'approche que nous choisirons aura une forte répercution sur notre éditeur, puis nous analyserons l'autre partie de la définition qui est la création et la transformation de dessins (3.2.2.) ensuite, nous introduirons la notion de famille de dessins (3.2.3.) et enfin, nous parlerons succintement de la création et transformation d'une famille de dessins (3.2.4.).

3.2.1. ANALYSE DESCRIPTIVE DE LA NOTION DE DESSIN

Nous allons d'abord, sur base de 2 manières différentes d'approcher la conception d'un dessin (3.2.1.1.), dégager les concepts de base liés à l'approche qui sera la nôtre (3.2.1.2.).

3.2.1.1. CONCEPTION D'UN DESSIN

Il existe différentes manières d'approcher la conception d'un dessin. Parmi elles, nous allons en développer deux plus particulièrement, la 2ème approche étant celle que nous avons choisie.

1) on peut concevoir un dessin à l'aide d'une gomme et d'un crayon et considérer ce dessin comme étant une suite de traits ou de points.
2) une idée plus complexe mais plus puissante est d'essayer de donner une structure au dessin.
   Pour cela, on peut voir un dessin comme étant un assemblage d'éléments ou de briques, chaque nouveau dessin devenant lui-même une nouvelle brique.
   Le dessinateur concevrait donc un nouveau dessin (nouvelle brique) à l'aide d'un certain nombre de briques existantes, parmi lesquelles on pourrait retrouver des briques dites de base (notion de combinaison de briques).
   Les briques de base seraient par exemple une droite
   un cercle
   un rectangle etc ...

Nous avons choisi de baser notre approche sur la 2ème approche énoncée car elle possède les avantages suivants { notion de structuration
notion de réutilisation.

Ces avantages apparaîtront clairement par la suite.
De cette approche, vont découler un certain nombre de concepts.

3.2.1.2. CONCEPTS

Nous allons ici développer un certain nombre de concepts qui seront les concepts de base de notre approche.
Ces concepts sont les suivants :

dessin (3.2.1.2.1.)
dessin constituant (3.2.1.2.2.)

3.2.1.2.1. Notion de dessin

Lorsqu'on observe un dessin, on peut ne pas le considérer comme étant une suite de traits ou de points.
On peut le découper en un ensemble d'entités, chaque entité pouvant être vue comme étant un dessin propre. Comme chaque entité est considérée comme un dessin, elle peut elle-même être découpée et ainsi de suite jusqu'au moment où on va rencontrer des entités que l'on va juger comme étant élémentaires, non décomposables. Ce serait le cas d'une droite, d'un cercle par exemple. On peut en déduire qu'un dessin est donc constitué d'un ensemble de sous-dessins (combinaison de dessins).

En se basant sur une telle approche, on peut considérer un dessin comme pouvant être de 2 types.

soit un dessin de base qui par définition est indécomposable.
Ces dessins de base (par ex. droite, cercle) constituent des entités élémentaires.
soit un dessin composé c'est-à-dire un dessin qui peut être décomposé en un certain nombre de dessins.

Sous une autre formulation, on obtient

dessin ::= dessin* /dessin base
dessin base ::= droite, cercle ...

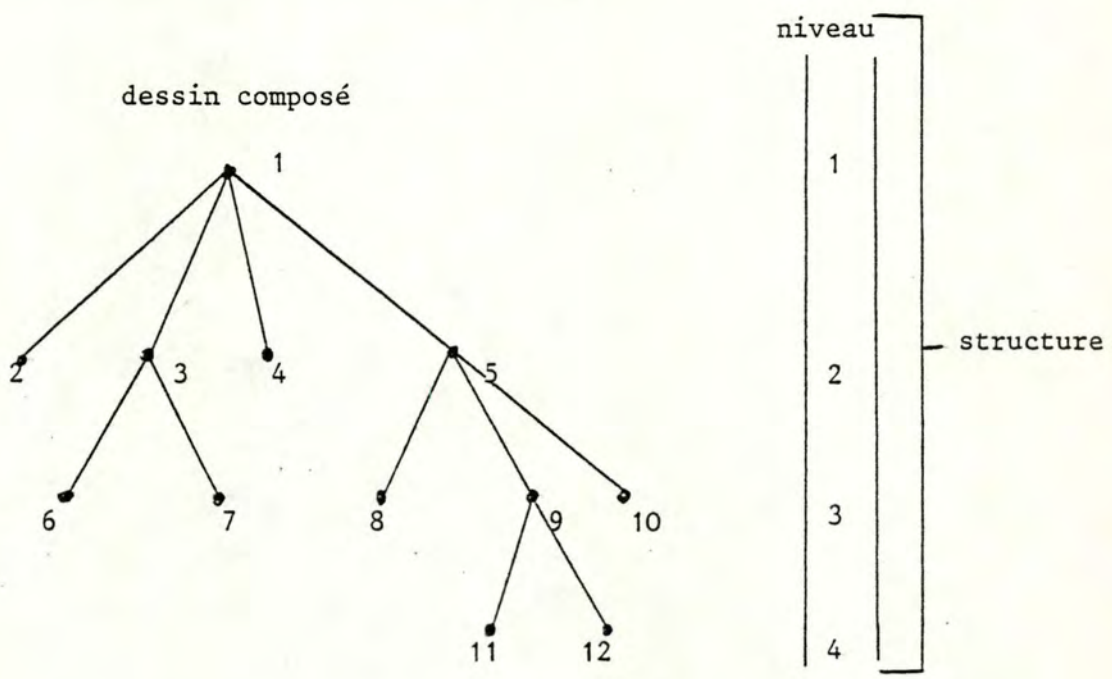
De cette définition vont découler les notions de structure et de dessin constituant.

3.2.1.2.2. Notions de structure et de dessin constituant

Suite à la définition donnée au point 3.2.1.2.1., on dispose pour un dessin composé d'une structure arborescente, chaque noeud de cette structure étant un dessin. Le dessin associé à chaque noeud (excepté la racine) de cette structure sera appelé dessin constituant, la racine étant elle, appelée dessin composé.

Une telle structure sera inexistante pour une dessin de base par définition (on peut dire que sa structure est un singleton ; uniquement la racine).

Le schéma de la structure d'un dessin composé sera donc le suivant :



Le dessin composé sera la racine de la structure.

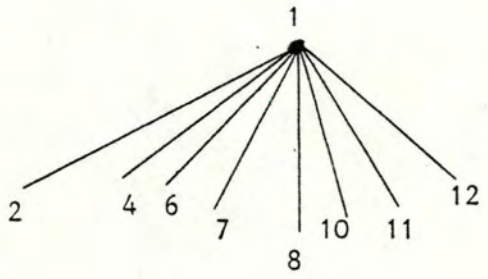
Les dessins constituants seront :

- des feuilles de cette structure, s'ils sont des dessins de base (noeuds : 2, 4, 6, 7, 8, 10, 11, 12)
- des noeuds (≠ feuilles de cette structure, s'ils sont des dessins composés (noeuds : 3, 5, 9).

L'intérêt d'une telle approche réside dans le fait que l'on va pouvoir - combiner des dessins

- manipuler ou agir sur des parties de dessins (les noeuds de la structure) constituant des entités plus riches que les entités simples, droite, cercle ... (ex. 3, 5, 9)

Dans une approche où la structure ne possède que 2 niveaux c'est-à-dire qu'on considère le dessin comme n'étant composé que par un ensemble d'entités de base, on perd cette possibilité



de combinaison de dessins et notre niveau d'action est considérablement réduit.

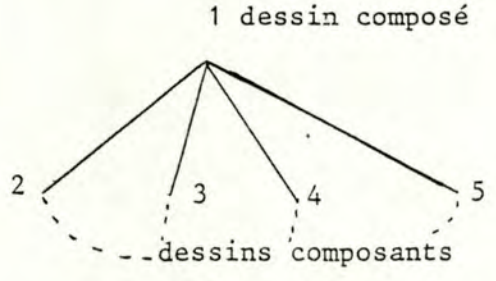
3.2.2. CREATION ET TRANSFORMATION D'UN DESSIN

Après avoir décrit l'approche de la notion de dessin qui sera la nôtre (notion centrale de la définition d'un éditeur graphique), nous allons analyser en quoi une telle approche peut être intéressante en ce qui concerne la 2ème partie de la définition : créer (3.2.2.1.) et transformer un dessin (3.2.2.2.).

3.2.2.1. CREATION D'UN DESSIN

La définition d'un dessin choisie au point 3.2.1.2.1. va influencer la création proprement dite.
L'utilisateur dispose donc d'un ensemble de briques de base (les dessins de base).
Une approche aurait été de créer un nouveau dessin (dessin composé) en combinant des dessins de base uniquement.
Une telle approche est assez limitative, car ne pouvant réutiliser les dessins composés ainsi créés, on ne peut enrichir le lot de briques. On est alors obligé de systématiquement tout construire à l'aide du plus bas niveau : les briques de base.
L'approche que nous avons choisie, nous permet par contre, de combiner en plus des dessins de base, des dessins composés afin d'en créer de nouveaux. Les dessins ainsi utilisés seront appelés des dessins composants. Ainsi, nous considérons les dessins composés comme étant des nouvelles briques disponibles. Il est important de remarquer qu'on ne doit pas nécessairement connaître la structure de ces dessins composants. On peut les combiner tels des briques opaques.
Dans ce cas là, pour un dessin ainsi créé, on peut donc limiter au moment de la création, notre connaissance de sa structure au 2ème niveau. Les autres niveaux pourront être déduits de la structure propre des dessins composants, bien que conceptuellement, ce soit toute la structure qui définisse le dessin.
Nous appellerons par la suite les 2 premiers niveaux de la structure dessin : sa sous-structure. Les noeuds du 2ème niveau seront appelés des dessins composants.

En reprenant l'exemple du point 3.2.1.2.2., la sous-structure est



Cela nous permet d'utiliser un système pour lequel il est seulement nécessaire de connaître l'ensemble des dessins existants (et pas comment ils sont constitués : leur structure).
En conclusion, on peut dire que notre approche d'un dessin, basée sur la structuration, nous permet d'élever notre niveau de création en terme de combinaison.

3.2.2.2. TRANSFORMATION D'UN DESSIN

Après avoir créé un dessin, il paraît indispensable de pouvoir agir dessus (action), le modifier (ajouter, retirer un composant).
Les actions vont être exprimées par ce que l'on va appeler des opérations.
Exemple : choisir une certaine couleur.
faire subir une rotation, une translation ...

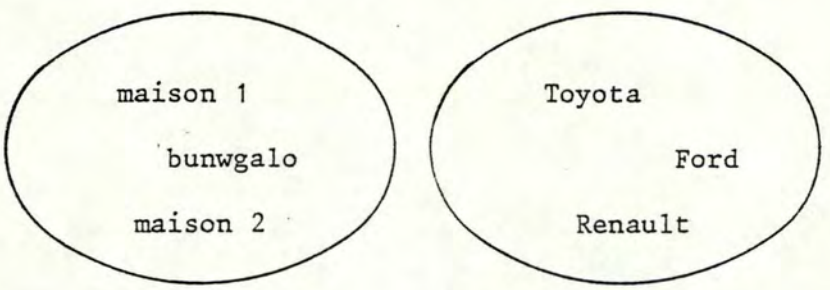
Une idée qui nous a paru assez puissante et dont les avantages apparaîtront clairement par la suite, était de ne pas limiter l'application des opérations à la seule sous-structure du dessin mais bien à tout dessin constituant de sa structure.
Cette idée implique bien sûr la nécessité de connaître celle-ci.
Au niveau de l'effet de l'opération, on peut dire qu'il portera sur le noeud de l'arbre de structure (associé au dessin constituant sur lequel porte l'opération) et sur le sous-arbre défini par ce noeud.

3.2.3. NOTION DE FAMILLE DE DESSINS

Lorsqu'on observe plusieurs dessins, il se peut que certains dessins se ressemblent, ils ont des caractéristiques communes. Cela ne veut pas dire pour autant qu'ils sont identiques.
Par définition, nous dirons que 2 dessins se ressemblent lorsqu'ils ont même structure d'arbre (définissant ainsi les caractéristiques communes).

De là, va découler l'idée de regrouper les dessins se ressemblant en une famille.
Les caractéristiques communes aux dessins se ressemblant définissent la famille de dessins, les caractéristiques différentes définissant un élément de cette famille (c'est-à-dire un dessin particulier).

exemple : famille des maisons famille des voitures



Le dessin "maison 1" est constitué des dessins : mur, toit, porte et fenêtre. Il en va de même pour maison 2 (même structure d'arbre). Les seules différences se situant, par exemple, au niveau de la taille de la maison, de la couleur de la porte.
Ces 2 dessins sont de la même famille (famille des maisons).
On peut tenir le même raisonnement concernant les voitures.

Comme il existe 2 types de dessins et que les dessins sont regroupés en familles, il existera 2 types de familles dépendant du type de dessins qu'elles regroupent

- des familles de base qui ne contiennent que des dessins de base
- des familles non de base qui ne contiennent que des dessins composés.

Cette notion de famille de dessins sera un des concepts fondamentaux de notre approche.

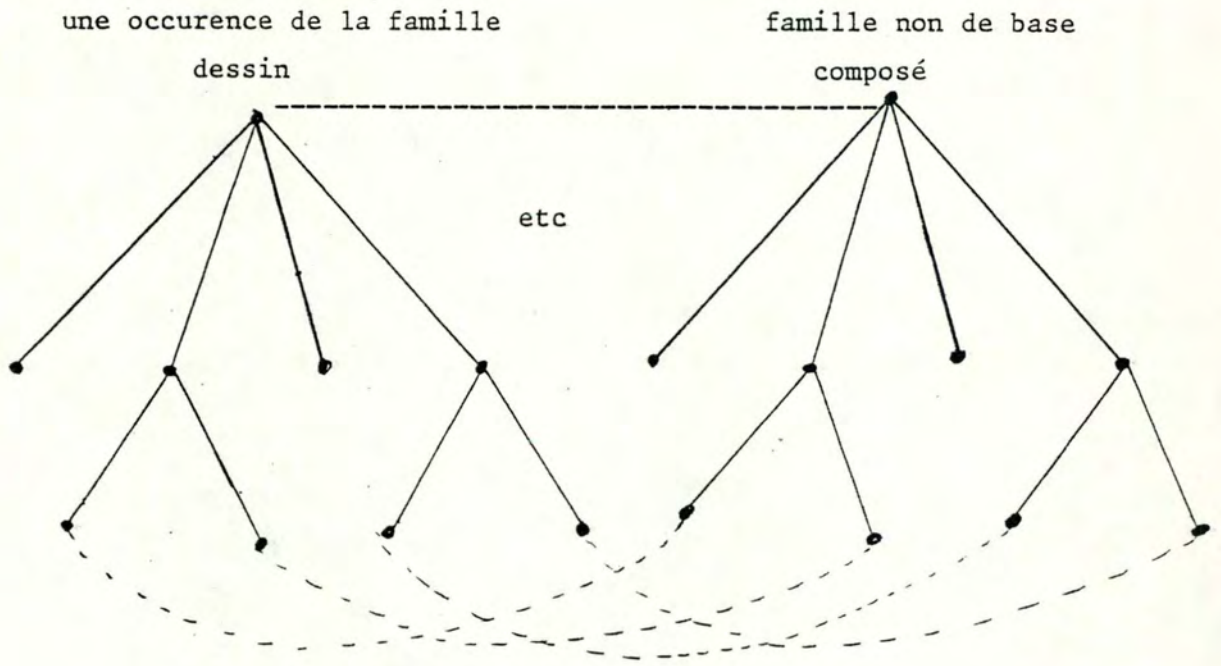
On peut, par exemple, représenter une famille de dessin par une fonction. Comme on sait que les caractéristiques différentes définissent les dessins de la famille, celles-ci sont exprimées par les paramètres de la fonction.

Une valeur particulière de chaque paramètre de la fonction définit alors une valeur de cette fonction, donc un dessin de la famille.

Comme chaque dessin possède une structure et que les dessins d'une même famille ont même structure, alors on peut déduire qu'il y aura une correspondance entre la structure de la famille et la structure de ses dessins.

Explicitons maintenant cette notion de structure d'une famille.
Il est utile de rappeler que la structure arborescente d'un dessin est composée d'un ensemble de noeuds, chaque noeud représentant un dessin c'est-à-dire un élément d'une famille de dessins.
Quant à la structure arborescente d'une famille de dessins, elle sera correspondante à celle de ses dessins ; les noeuds correspondant un à un.

Le schéma de la structure d'une famille sera donc le suivant :



Si on considère un noeud de la structure du dessin, le noeud lui correspondant dans la structure de la famille sera lié à la famille de laquelle est issu ce dessin.

Un noeud (sauf la racine) représentera en fait

- soit tous les éléments d'une famille
- soit un sous-ensemble (non vide) de ces éléments

et sera appelé un constituant.

La racine sera appelée "composé" (explications cfr. 3.2.4.1.).

Il est aisé de déduire que la structure d'une famille de base est un singleton.

3.2.4. CREATION ET TRANSFORMATION D'UNE FAMILLE DE DESSINS

Après avoir décrit la notion de famille de dessins, nous allons analyser ensuite la création (3.2.4.1.) et la transformation de ces familles de dessins (3.2.4.2.).

3.2.4.1. CREATION D'UNE FAMILLE DE DESSINS

Pour créer une nouvelle famille de dessins, nous allons nous servir de familles existantes.
Donc, en parallèle à la notion de combinaison de dessins, nous aurons également ici, la même notion (combinaison de familles).
Pour chaque famille utilisée, nous pourrons :

- soit laisser variables toutes les caractéristiques différentes et ainsi désigner tous les éléments de la famille (1)
- soit laisser variables seulement un certain nombre de caractéristiques différentes et ainsi désigner un sous-ensemble de plus d'un élément de la famille (2)
- soit ne laisser variable aucune des caractéristiques différentes et ainsi désigner un élément de la famille (un dessin) (3).

L'ensemble des familles utilisées ainsi que la manière dont on s'est servi de chacune d'elle (soit (1), (2) et (3)), constitueront la définition de la nouvelle famille.

Pour obtenir un élément de cette nouvelle famille ainsi créée, nous donnerons une valeur à chaque caractéristique laissée variable.

Dans la structure de la nouvelle famille, chaque constituant du 2ème niveau représentera soit tous les éléments d'une famille choisie
soit un sous-ensemble de plus d'un élément d'une famille choisie
soit un élément d'une famille choisie.

Nous appellerons donc par la suite les 2 premiers niveaux de la structure d'une famille sa sous-structure. Chaque noeud de cette sous-structure sera appelé un composant, sauf la racine qui est appelée composé.
Au niveau de la définition d'une famille, nous pouvons nous limiter à une telle définition car les familles associées à chaque composant possèdent également une telle définition etc ...

3.2.4.2. TRANSFORMATION D'UNE FAMILLE DE DESSINS

Après avoir créé une famille de dessins, il paraît intéressant de pouvoir la modifier.
Modifier une famille reviendra à aller modifier la définition de celle-ci.
On pourra
- retirer un noeud de sa sous-structure (ce qui implique la disparition dans sa structure du sous-arbre défini par ce noeud)
- ajouter un noeud à sa sous-structure (ce qui revient à ajouter à sa structure, la structure de la famille associée à ce nouveau noeud)
- modifier la manière dont on s'était servi d'une famille (cfr. (1), (2) et (3))
- appliquer une opération à un constituant (idem (3.2.2.2.)).

Cependant, il est important de se rendre compte qu'une modification apportée à la définition d'une famille peut avoir de multiples répercutions sur d'autres familles. Ce serait le cas, si on s'était déjà servi de la famille que l'on vient de modifier dans la définition d'autres familles. Un certain nombre de choix seront fait ultérieurement en ce qui concerne ce problème (cfr. 3.4.6.2.1.).

3.3. REPRESENTATION DE CES CONCEPTS DE BASE

Après avoir décrit les concepts de base que nous allons mettre en oeuvre dans notre éditeur graphique :

- notion de dessin (3.2.1.) et (3.2.2.)
- notion de famille de dessins (3.2.3.) et (3.2.4.),

nous allons introduire la notion d'opérateur (3.3.1.) qui va nous permettre de décrire une famille de dessins, ainsi que le langage LG (3.3.2.) qui sera le langage dans lequel on écrira un opérateur.

3.3.1. NOTION D'OPERATEUR

Nous allons dans un premier temps, donner un certain nombre de définitions liées à la notion d'opérateur (3.3.1.1.) et ensuite introduire un concept tout à fait particulier à l'approche que nous avons choisie : les modificateurs (3.3.1.2.).

3.3.1.1. DEFINITIONS ET CONCEPTS LIES A LA NOTION D'OPERATEUR

Il nous fallait un moyen pour décrire une famille de dessins. Nous avons utilisé la notion d'OPERATEUR.

Définition : un opérateur décrit une famille de dessins.

Comme il existe 2 types de familles et que les familles sont définies par des opérateurs, il existera 2 types d'opérateurs :

- les opérateurs de base décrivant les familles de base
- les opérateurs composés décrivant les familles non de base.

Les opérateurs de base sont des opérateurs prédéfinis. Ils constituent les matériaux de base de l'utilisateur.

ex. : opérateur de base droite, rectangle, cercle, ...

On trouvera une liste de ces opérateurs de base en annexe D.

Les opérateurs composés sont des opérateurs définis par l'utilisateur à l'aide d'opérateurs de base et/ou d'opérateurs composés créés par lui précédemment.

Ils sont écrits en langage LG (cfr. 3.3.2.).

De la définition d'un opérateur vont découler, pour un opérateur, les notions suivantes : structure, sous-structure, composé, composant et constituant.

La structure de l'opérateur correspondra à la structure de la famille décrite par cet opérateur. Il en sera de même pour les 4 autres notions.

Abordons maintenant la description d'un opérateur.

Comme indiqué précédemment, un opérateur composé sera écrit en langage LG (cfr. 3.3.2.). Il possédera une structure. Cette structure sera limitée à un singleton pour un opérateur de base.

Un opérateur composé sera constitué de 3 parties :

1) son nom
2) une liste de paramètres
3) un corps (qui en langage LG sera appelé "bloc").

Par contre, un opérateur de base, par définition, ne possédera pas de corps mais uniquement les parties 1 et 2.

En ce qui concerne la liste de paramètres :

- celle-ci peut être vide, ce qui indiquerait que la famille de dessins décrite par cet opérateur ne contiendrait qu'un seul dessin. On pourra bien sûr se servir plusieurs fois de ce seul dessin.
- si cette liste est non vide, alors cette liste représente l'ensemble des caractéristiques différentes des éléments de la famille. Elle constituera un moyen qui va nous permettre d'atteindre un élément particulier de la famille. Selon l'exemple du point 3.2.3., les paramètres seront en fait les variables de la fonction.
  En donnant une valeur à chacun des paramètres, on définit un élément particulier de la famille.

En ce qui concerne le corps :

Le corps de l'opérateur va décrire les caractéristiques communes aux dessins de la famille définissant celle-ci.
Le corps contiendra, entre autre, la manière dont on s'est servi des opérateurs.

Afin de créer un nouvel opérateur, nous allons utiliser des opérateurs existants.
Comme au niveau de la définition d'une famille, on limite la définition à la manière dont on s'est servi des familles, la définition de l'opérateur se limitera donc à décrire la manière dont on s'est servi des opérateurs (combinaison d'opérateurs).

Remarque :

Dans le cadre du travail que nous avons développé, nous nous sommes volontairement limités.
L'utilisation de l'opérateur dans sa propre définition est interdite car un opérateur étant écrit en langage LG et ce langage simplifié ne possédant pas de structure de contrôle, il aurait été impossible de gérer la réussivité et de savoir quand s'arrêter.

Il nous est ensuite nécessaire d'introduire une manière de désigner les noeuds d'une structure.

- structure d'un dessin :

  Conceptuellement, nous avons vu qu'à un noeud de la structure est associé un dessin.

  Le noeud possédera un nom, tandis que le dessin associé au noeud aura comme nom la suite des noms des noeuds constituant le chemin allant, de la racine de la structure au noeud associé à ce dessin.

  Il est évident que cette suite de noms désignera un dessin uniquement lorsqu'il se trouve dans la structure d'un autre.

- structure d'une famille de dessins :

  Suite au parallélisme entre la structure d'une famille et la structure de ses dessins, expliquée précédemment (cfr. 3.2.3.), on dira que les noeuds correspondant des 2 structures auront même nom, excepté pour les racines.

  La racine de la structure de la famille aura pour nom, le nom de la famille et celle de la structure d'un dessin, le nom de ce dessin issu de la famille.

  Tous les dessins d'une famille auront des noms différents. De plus, dans l'ensemble du système (notre éditeur graphique), tous les dessins posséderont des noms différents.

- structure d'un opérateur :

  Il y aura identité complète entre les noms des noeuds correspondant des 2 structures (celle de la famille et celle de l'opérateur représentant cette famille).

Après cette explication concernant les noms associés aux noeuds des différentes structures, reprenons maintenant la suite de l'analyse de la définition d'un opérateur.

A chaque composant de la structure de l'opérateur, correspondra dans le corps de l'opérateur une instruction opérateur.

Cette instruction contiendra :

- le nom du noeud associé au composant : qualificateur
- le nom de l'opérateur utilisé ainsi que la manière dont on l'utilise : nom opérateur, liste de paramètres effectifs
- les modificateurs éventuels associés à cette utilisation (cfr. 3.3.1.2.).

Le paramétrage des opérateurs est quelque chose d'assez simple mais qui peut être compliqué pour notre type d'utilisateur.
Par exemple, il n'est pas toujours facile de déterminer les coordonnées exactes d'un point.
C'est donc pour cela que nous avons introduit la possibilité de calculer des valeurs de paramètres via ce qui sera appelé en langage LG, des instructions d'affectation.
D'autre part, pour faciliter le travail de l'utilisateur, nous avons défini dans le langage LG des instructions d'état permettant de choisir pour la couleur de fond de l'écran, le type du trait utilisé pour tracer les dessins de la famille, la couleur du trait ; une autre valeur que la valeur attribuée par défaut à tous les opérateurs.
On peut trouver la liste des états disponibles en annexe F, ainsi qu'une présentation de ceux-ci au point 3.4.5.3.

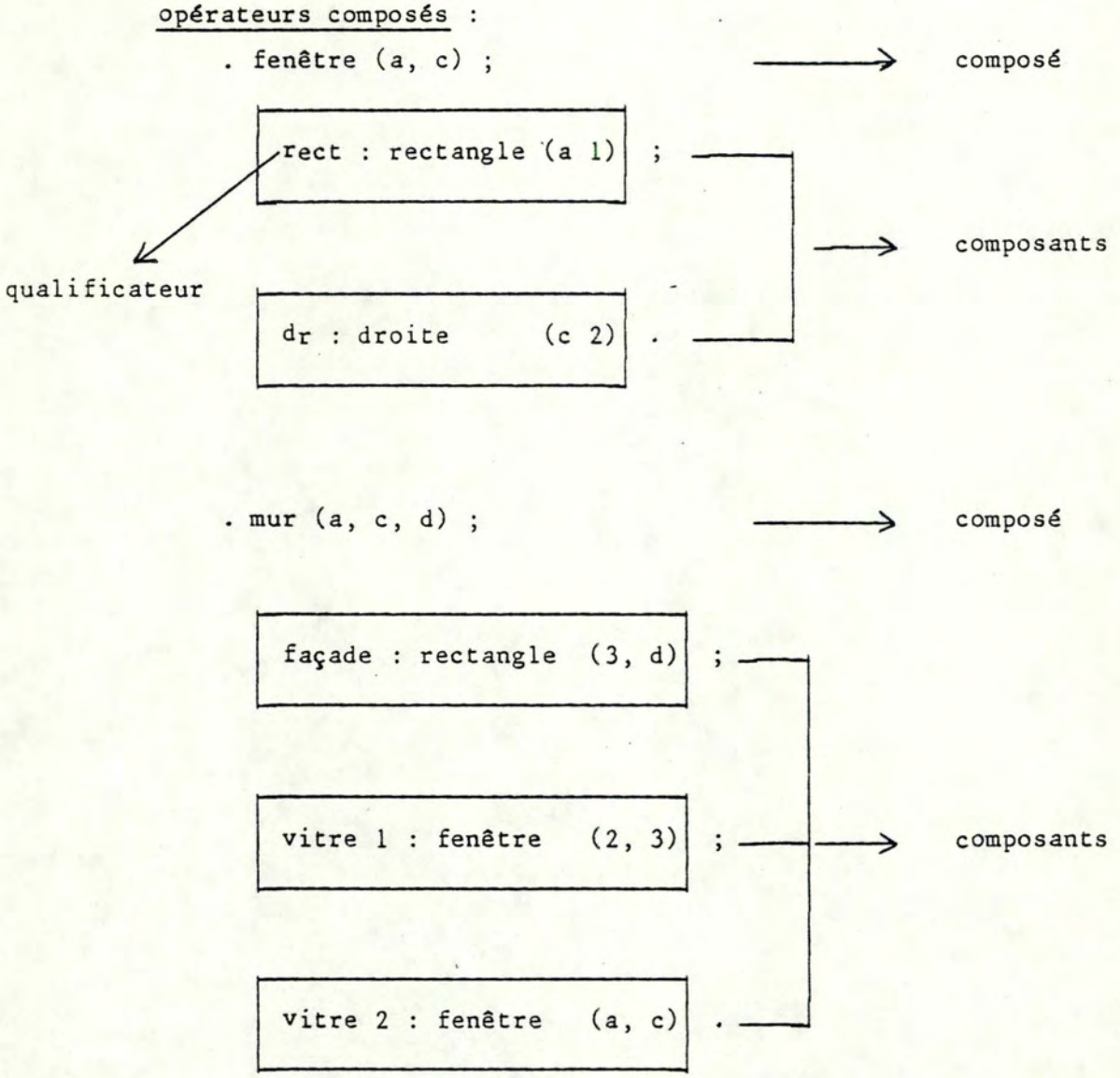
En conclusion, on peut dire que le bloc d'un opérateur contiendra

des instructions opérateur
des instructions état
des instructions d'affectation.

Pour terminer ces définitions et concepts liés à la notion d'opérateur, donnons un exemple de la définition et de la structure d'un opérateur (exemple (1)) :

<u>opérateurs de base</u> :

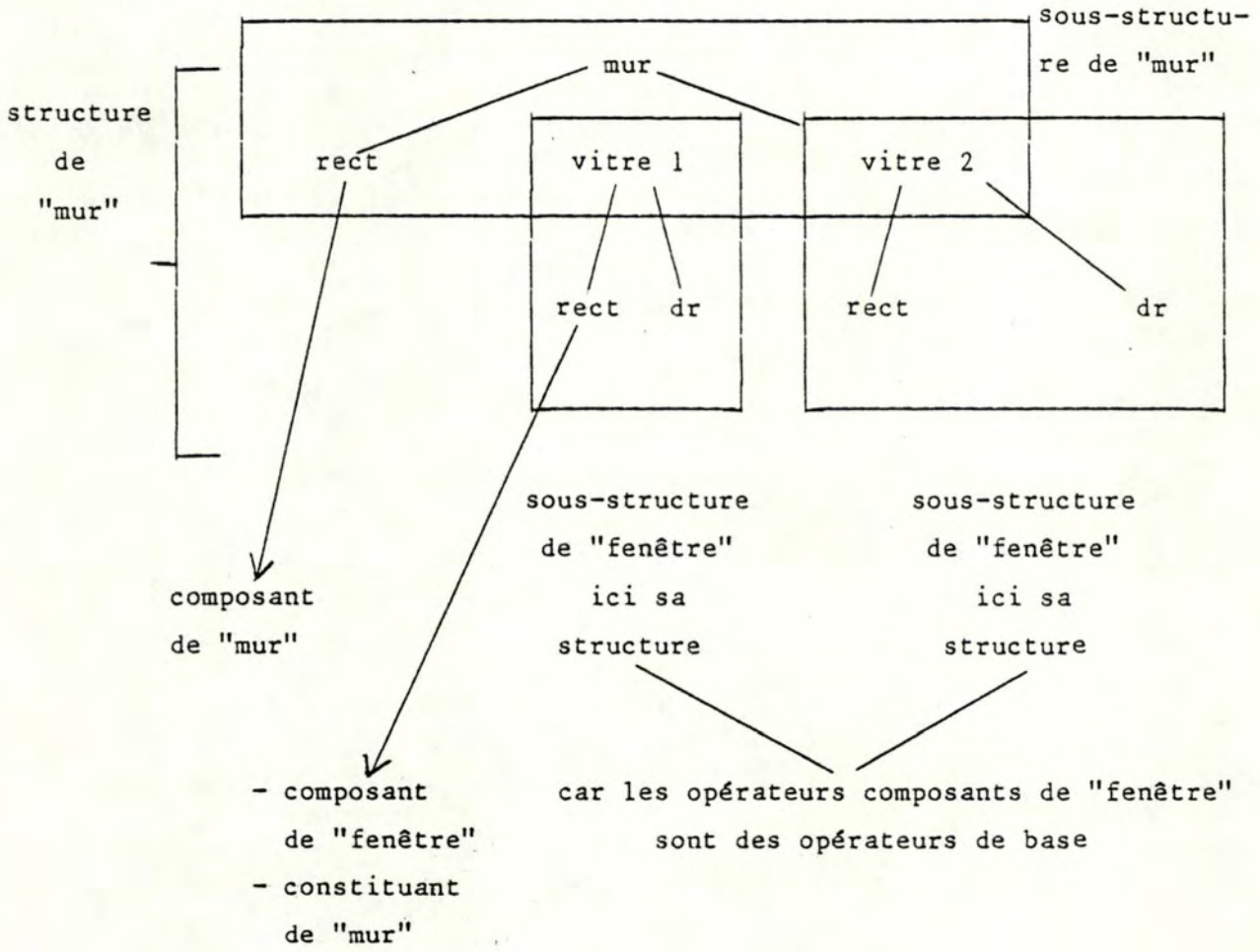
. droite (a, b)
. rectangle (c, d)
. cercle (h, g)



remarque

Comme aucune instruction d'état n'a été mentionnée dans le texte des opérateurs composés "fenêtre" et "mur" la couleur de fond de l'écran, le type et la couleur du trait utilisés pour tracer les dessins de ces 2 familles auront pour valeur : les valeurs par défaut attribuées par le système.

la structure de l'opérateur "mur" sera la suivante :



En fait, "fenêtre" est le nom de l'opérateur dont on s'est servi dans la définition de "mur".
Cela définit dans la structure de "mur", 2 composants (car on s'est servi 2 fois de l'opérateur "fenêtre") dont les noms sont "mur - vitre 1" et "mur - vitre 2".

3.3.1.2. LES MODIFICATEURS

Nous allons maintenant développer une idée qui sera fondamentale au niveau de nòtre approche : les modificateurs.
Nous allons d'abord présenter une introduction à la notion de modificateur (3.3.1.2.1.), puis parler de la notion de famille implicite qui découle de cette notion de modificateur (3.3.1.2.2.) ensuite critiquer cette notion de modificateurs selon différents points de vue (3.3.1.2.3.) et enfin traiter le problème de l'héritage de ces modificateurs (3.3.1.2.4.).

3.3.1.2.1. Introduction

Afin d'introduire cette notion de modificateur, considérons l'arbre de structure d'un opérateur.
Comme expliqué précédemment (3.3.1.1.), seuls les 2 premiers niveaux de la structure nous intéressent dans la définition d'un opérateur (bien que conceptuellement toute la structure soit nécessaire).
Supposons alors que l'utilisateur désire effectuer une opération sur un constituant quelconque de cette structure (donc de niveau $\geq$ 2).
Que peut-il faire ?

Nous avons pour cela introduit la notion de modificateur, qui va nous permettre d'exprimer de telles opérations.

Nous savons qu'à un composant (niveau 2) de cette structure, correspond une instruction opérateur.
De plus, un seul des composants définit un sous-arbre contenant le constituant choisi.

L'utilisateur indiquera alors dans l'instruction opérateur correspondant à ce composant, un modificateur contenant

- l'opération choisie
- une désignation du constituant sur lequel elle porte.

La désignation du constituant sera faite de la manière suivante :

on retiendra les noms des noeuds du chemin allant du composant définissant le sous-arbre au constituant choisi.

Pour cela, il est nécessaire de ne pas avoir une vue tronquée (les 2 premiers niveaux) mais bien une vue globale de toute la structure.

La liste (suite de noms) sera donc une suite de qualificateurs.

3.3.1.2.2. Notion de famille "implicite"

Le système tel que nous l'avons proposé, nous permet actuellement d'utiliser les opérateurs existants dont on ne doit pas connaître la structure mais seulement la liste de paramètres.

Le seul choix possible se situant au niveau des valeurs attribuées à ces paramètres.

Cependant, si on désire ajouter à l'opérateur une caractéristique non présente dans sa définition, on va devoir modifier celle-ci ou alors, ce que nous choisissons de faire, définir un autre opérateur calqué sur celui que l'on avait.

Afin de permettre cela, nous avons introduit le concept de modificateur.

Reprenons l'exemple (1) (cfr. point 3.3.1.1.).

Si pour le composant "vitre 1" du composé "mur", aucun paramètre n'exprime la couleur, alors il est possible d'ajouter l'opération couleur (cfr. point 3.2.4.2.) sous forme d'un modificateur :

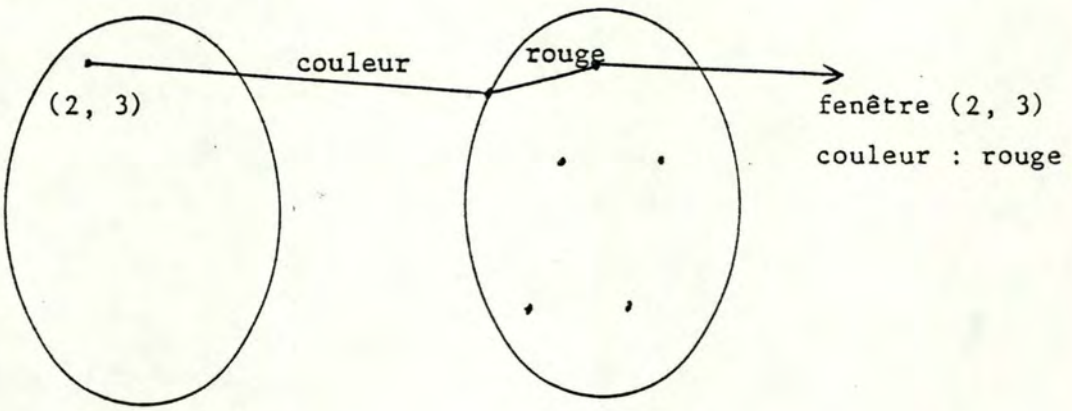
mur (a, c, d) ;
façade : rectangle (3, d) ;
vitre 1 : fenêtre (2, 3) [couleur : rouge] ;
modificateur
vitre 2 : fenêtre (a, c).

(ceci constitue l'exemple (2))

Les modificateurs vont nous permettre de décrire ces opérations.
La liste des opérations se trouve en annexe E.
Leur présentation sera faite au point (3.4.5.2.).
Revenons maintenant à l'exemple (2).
L'opération couleur, telle qu'elle a été utilisée dans cet exemple, porte sur "fenêtre (2, 3)". En fait, fenêtre (2, 3) et couleur définissent implicitement un nouvel opérateur (nouvelle famille).

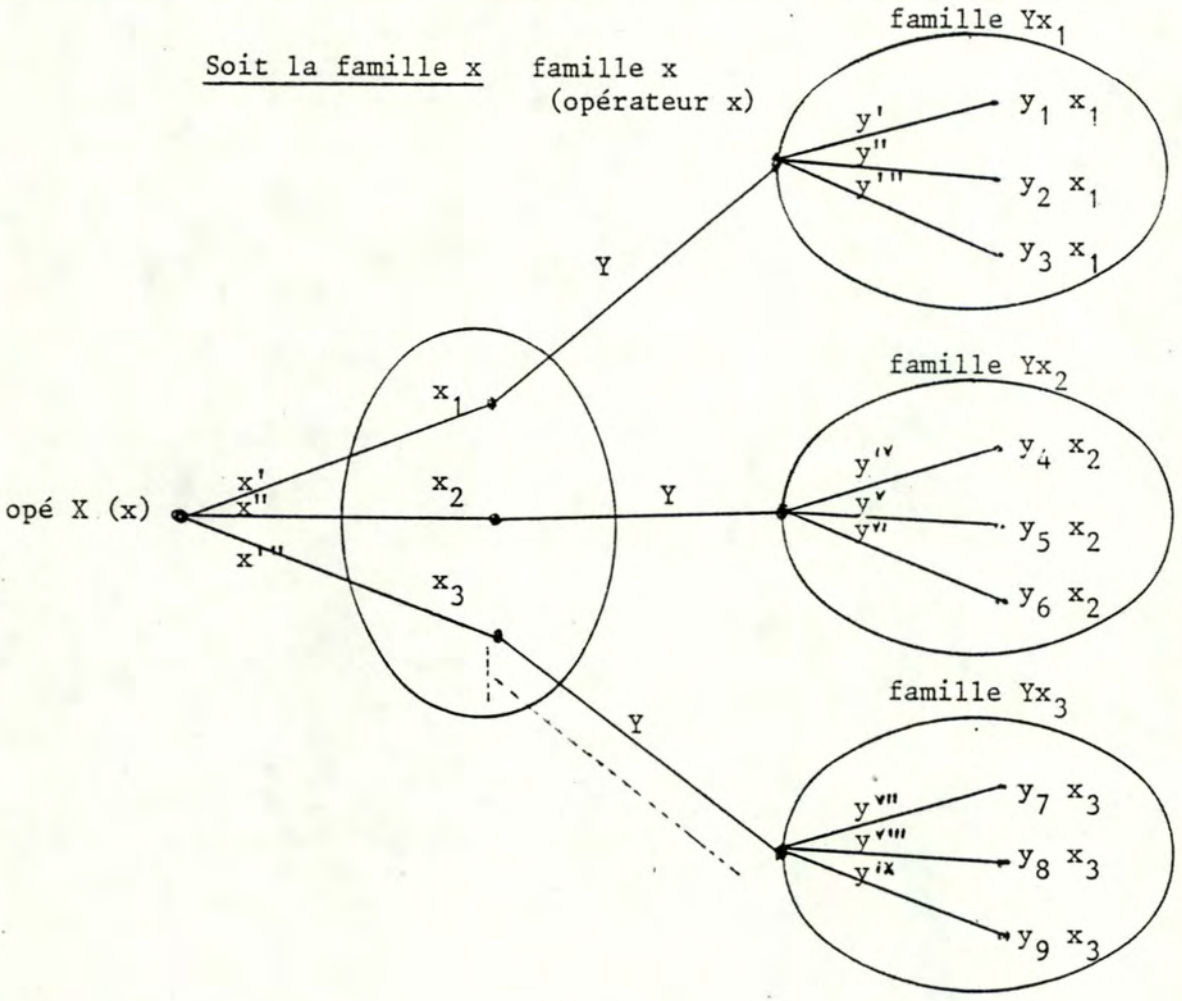
famille fenêtre (x, y)

famille fenêtre (2, 3) + couleur



Cette nouvelle famille est "implicite", on ne lui donne pas un nom particulier.
Donc, on peut dire que le modificateur est un moyen pour, ayant une famille, en construire une autre.
Cependant, on ne peut pas la manipuler par la suite ; c'est-à-dire l'utiliser lors de la création d'une nouvelle famille.
C'est en ce sens que l'on dit que la famille est "implicite".

Généralisons maintenant cette notion de famille implicite :



avec les significations suivantes

. X représente le nom de la famille

. X (x)
  |
  liste paramètres formels de X

. x_i représente un élément de cette famille, pour $1 \leq i \leq n$ avec n non fixé

. Y un modificateur

. x' = une liste de paramètres actuels de X

. y' = une valeur de Y

. $y_i\ x_i$ représente un élément de la famille yx_i, pour $1 \leq j \leq n$ avec n non fixé.

On peut alors synthétiser cela, en disant que :

opér X (x)	définit la famille X
opér X (x ')	définit un élément de la famille X c'est-à-dire le dessin x_1
opér X (x ') Y	définit la famille Yx_1
opér X (x ') y '	définit un élément de la famille Yx_1 c'est-à-dire le dessin $y_1 x_1$

Reprenons maintenant notre analyse de l'exemple (2).
L'opération couleur porte sur une utilisation de l'opérateur "fenêtre" : (fenêtre (2, 3)).
Le composant de "mur" défini par cette utilisation et cette opération, porte le nom : "mur-vitre 1" et est lié à un opérateur implicite : fenêtre (2, 3) couleur.
Outre l'utilisation de "fenêtre", cette opération influencera le sous-arbre ayant pour racine le composant "mur_vitre 1".
(cfr. règles d'héritage point 3.3.1.2.4.).

Comme cette opération couleur porte sur l'utilisation de "fenêtre" et que cette utilisation se trouve dans l'instruction opérateur correspondant à un composant (2ème niveau : "mur_vitre 1"), le modificateur contenant cette opération sera dit global.

Si dans la définition de "mur", une opération porte sur un constituant de la structure de "mur" (niv > 2), elle sera contenue dans un modificateur dit local. Il sera indiqué dans l'instruction opérateur correspondant au composant (noeud), définissant le sous-arbre contenant le constituant.

Dans l'instruction opérateur, il pourra y avoir un modificateur local par constituant, appartenant au sous-arbre défini par le composant correspondant à cette instruction opérateur (liste modificateurs locaux). Le modificateur local d'un constituant sera présent à la condition qu'il y ait au moins une opération le concernant et on y regroupera toutes les opérations le concernant.

L'ensemble des modificateurs globaux et locaux d'un composant sera appelé <u>modificateur</u> et apparaîtra dans cet ordre.

Pour terminer, formalisons cette notion à l'aide de la notation B.N.F. (Backus-Naur Form) qui sera utilisée dans le langage LG.

Une éventuelle répétition d'une forme (0 à n fois) est indiquée en la mettant entre accolades { et } .

< modificateur > ::= < liste de modificateurs globaux >
< liste de modificateurs locaux >

< liste de modificateurs globaux > ::= [< modificateur global > {, < modificateur global >}] / < vide >

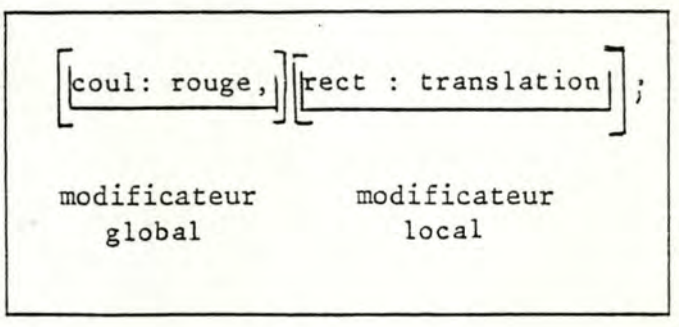
< liste de modificateurs locaux > ::= [< modificateur local > {, < modificateur local >}] / < vide >

< modificateur global > ::= < opération >

< modificateur local > ::= < liste de qualificateurs d'opérateur >
: < opération > {, < opération >}

Exemple (3) :

mur (a, c, d) ;
façade : rectangle (3, d) ;
vitre 1 : fenêtre (2, 3)



modificateur

vitre 2 : fenêtre (a, c).

3.3.1.2.3. Critique de cette notion de modificateurs

Nous allons critiquer cette notion selon trois points de vue différents : a) au point de vue utilisateur
b) au point de vue méthodologique
c) au point de vue conceptuel

a) Au point de vue utilisateur

- Un modificateur est un outil permettant une grande liberté d'action sur un constituant, sans modifier la définition de l'opérateur utilisé. Cela a comme conséquence de ne pas influencer les utilisations ultérieures de celui-ci.
- Cela permet également, en cas d'oubli lors de la définition, de ne pas devoir recommencer tout le travail.

b) Au point de vue méthodologique

A ce point de vue, cet outil est plus critiquable car une bonne méthode de travail nous précise : qu'il faut penser de préférence à tout, avant de définir un opérateur.

c) Au point de vue conceptuel

A ce point de vue, cet outil nous permet de nous élever à un niveau supérieur.
Je dispose d'opérateurs (≈fonctions) qui définissent en fait des familles de dessins.
Une première étape était le passage de la notion de dessin (1er niveau) à la notion de famille de dessins (2ème niveau).
Cet outil nous permet de passer du 2ème niveau à un 3ème niveau qui est la notion de famille implicite, permettant de ne pas modifier la définition de la famille, tout en l'adaptant à l'utilisation désirée.
Cela nous apporte au niveau méthodolotique, un outil qui peut être dangereux mais qui est intéressant et assez puissant.

3.3.1.2.4. L'héritage des modificateurs

A) Introduction

Comme indiqué en fin du point 3.3.1.2.2., en plus d'agir sur l'utilisation d'un opérateur, contenue dans une instruction opérateur (qui est associé au constituant sur lequel il porte), un modificateur, qu'il soit global ou local, aura une répercution sur le sous-arbre défini par ce constituant.

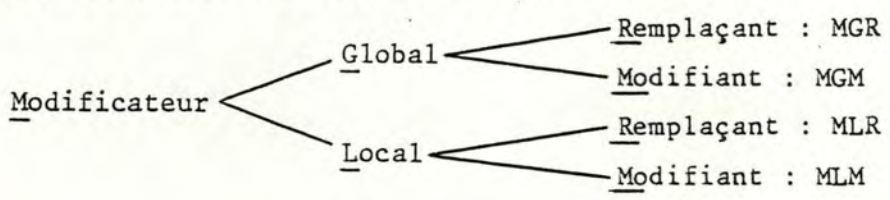
De ce fait, l'héritage des modificateurs consiste à déterminer quelle(s) opération(s) associer à l'utilisation d'un opérateur.
Les opérations relatives à l'utilisation d'un opérateur se trouvent donc dans ce que l'on a appelé les modificateurs.
Cependant, dans certaines commandes de l'éditeur, des opérations seront également présentes sous forme de modificateurs locaux et globaux.
Bien que la partie éditeur graphique (3.4.) ne soit pas encore abordée, nous ferons référence à certaines des commandes de l'éditeur, nous intéressant au niveau de l'analyse de ce point.
(car elles peuvent contenir des modificateurs).

B) Opérations Transférables/non transférables

Nous allons analyser le problème des différents types d'opérations.
Chaque opération est en fait

. soit transférable (Modifiant : M)
. soit non transférable (Remplaçant : R).

Donc, on obtient la synthèse suivante :



Les opérations transférables sont : rotation
translation
symétrie
réduction
dilatation.

Les opérations non transférables sont : couleur et trait.

On traitera l'instruction état ECRAN de manière particulière. On aurait pu considérer cet état comme étant une opération non transférable. Mais comme on désirait particulariser son traitement, nous l'avons analysée séparément (point G).

C) Occurences d'opération

Pour une opération donnée, une occurence de cette opération sera constituée - du nom de l'opération
- des valeurs des paramètres de cette opération.

On peut trouver une occurence des opérations rotation, translation, symétrie, réduction et dilatation dans les éléments suivants :

. dans les commandes de type "opération" et la commande "créerdessin" de l'éditeur (E1)
. dans un modificateur global (E2)
. dans un modificateur local (E3).

On peut trouver une occurence des opérations couleur et trait dans les éléments suivants :

. dans les commandes de type "opération" et la commande "créerdessin" de l'éditeur (E1)
. dans un modificateur global (E2)
. dans un modificateur local (E3)
. dans une instruction état (état COULEUR ou TRAIT) (E4).

Précisons que pour la commande "créerdessin" de l'éditeur, les opérations se retrouveront dans un modificateur global et/ou local.

D) Convention de vocabulaire

On sait qu'un modificateur est contenu

. dans une commande

. dans une instruction opérateur.

Si l'instruction opérateur qui le contient, correspond à un constituant de niveau i, alors on dira que le modificateur est de niveau i. Si le modificateur est contenu dans une commande, on dira qu'il est de niveau 1.

E) Occurences d'opérations relatives à l'utilisation d'un opérateur

On peut utiliser un opérateur par l'intermédiaire :

. d'une commande de l'éditeur (2 seulement nous intéressent : les commandes de type "opération" et la commande "créerdessin")

. d'une instruction opérateur.

Or une instruction opérateur correspond à un constituant de la structure d'un opérateur.

Par facilité de langage, on dira que les occurences d'opérations relatives à l'utilisation d'un opérateur (contenues dans une instruction opérateur) sont relatives au constituant correspondant à cette instruction opérateur.

Similairement, dans le cas d'opérations contenues dans les 2 commandes, on dira qu'elles sont relatives au composé (racine de la structure de l'opérateur concerné par ces commandes).

Soit la structure d'un opérateur (n niveaux)

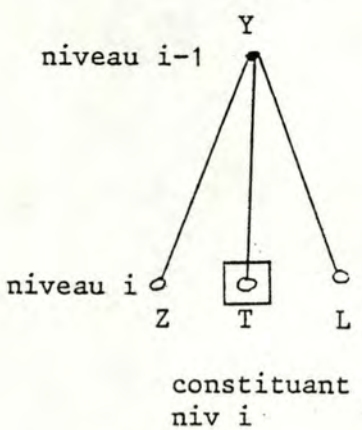
a) pour un composé les seules occurences possibles sont celles contenues dans une des 2 commandes.
(niv 1)
Les modificateurs apparaissant dans ces commandes seront associés à la racine de la structure (si ∃).

b) pour un constituant de niv $1 < i \leq n$

les occurences sont celles :

1. - relatives au constituant de niv i - 1 (composé si i = 2) lié au constituant de niv i par un chemin.
2. - contenues dans les modificateurs locaux de niv j (si ∃) ($1 \leq j < i$) concernant ce constituant.
3. - contenues dans le modificateur global de niv i (si ∃)
4. - contenues dans les dernières instructions d'état TRAIT et COULEUR se trouvant, dans le corps de l'opérateur utilisé dans l'instruction opérateur correspondant au constituant du niveau i - 1 (lié au constituant de niv i par un chemin), avant l'instruction opérateur correspondant au constituant.

exemple :



instr opérateur correspondant à Y

Y : X (a, b) modif.

définition de X

X (a, b)

TRAIT (———)
COULEUR (———)
Z : W (c, d) modif
COULEUR (———) annule la COULEUR précédente
T : V (e, f) modif
L : M (g, h) modif

pour T constituant de niv i, il faut tenir compte du TRAIT et de COULEUR entourés se trouvant dans le corps de l'opérateur X, utilisé dans l'instruction opérateur correspondant au constituant Y de niv i - 1.

F) Effet des occurences d'opérations relatives à l'utilisation d'un opérateur (composé ou constituant : convention de langage)

Toutes les occurences relatives à l'utilisation d'un opérateur (composé ou constituant , par convention de langage) vont avoir un effet sur celui-ci.

=Pour les opérations transférables :

cet effet se traduira par un produit de composition des effets de chacune des occurences des différentes opérations.

Dans le produit de composition, on peut retrouver

- des occurences d'opérations différentes
- plusieurs occurences de la même opération.

=Pour les opérations non transférables :

il faut prendre séparément chaque opération et pour chacune d'elle, il faut déterminer l'occurence la plus prioritaire.

(ce qui n'était pas le cas pour les transférables. Dans le produit de composition, on peut retrouver différentes opérations ; il ne faut pas les traiter toutes séparément).

LES OPERATIONS TRANSFERABLES

Lors de la constitution du produit de composition, il faut tenir compte d'une règle suivante :

exemple : Village (——)

M : maison (——) [translation A] [toit : rotation B]

Maison (——)

toit : toit (——) [translation F]

Pour le "toit", le produit de composition doit valoir

translation A (rotation B (translation F))

Il faut donc pour cela, lors de la construction, tenir compte du local (rotation B) avant le global (translation F). Le sens de construction est ——→ .

Par contre, le sens d'exécution est ←—— .

Si plusieurs locaux concernent le "toit", il faut d'abord indiquer le local provenant du niveau le plus élevé et ainsi de suite.

Dans un modificateur local, l'ordre de construction sera l'ordre d'apparition dans le modificateur local.

ex. : si on ajoute une symétrie dans le modificateur local "toit" :

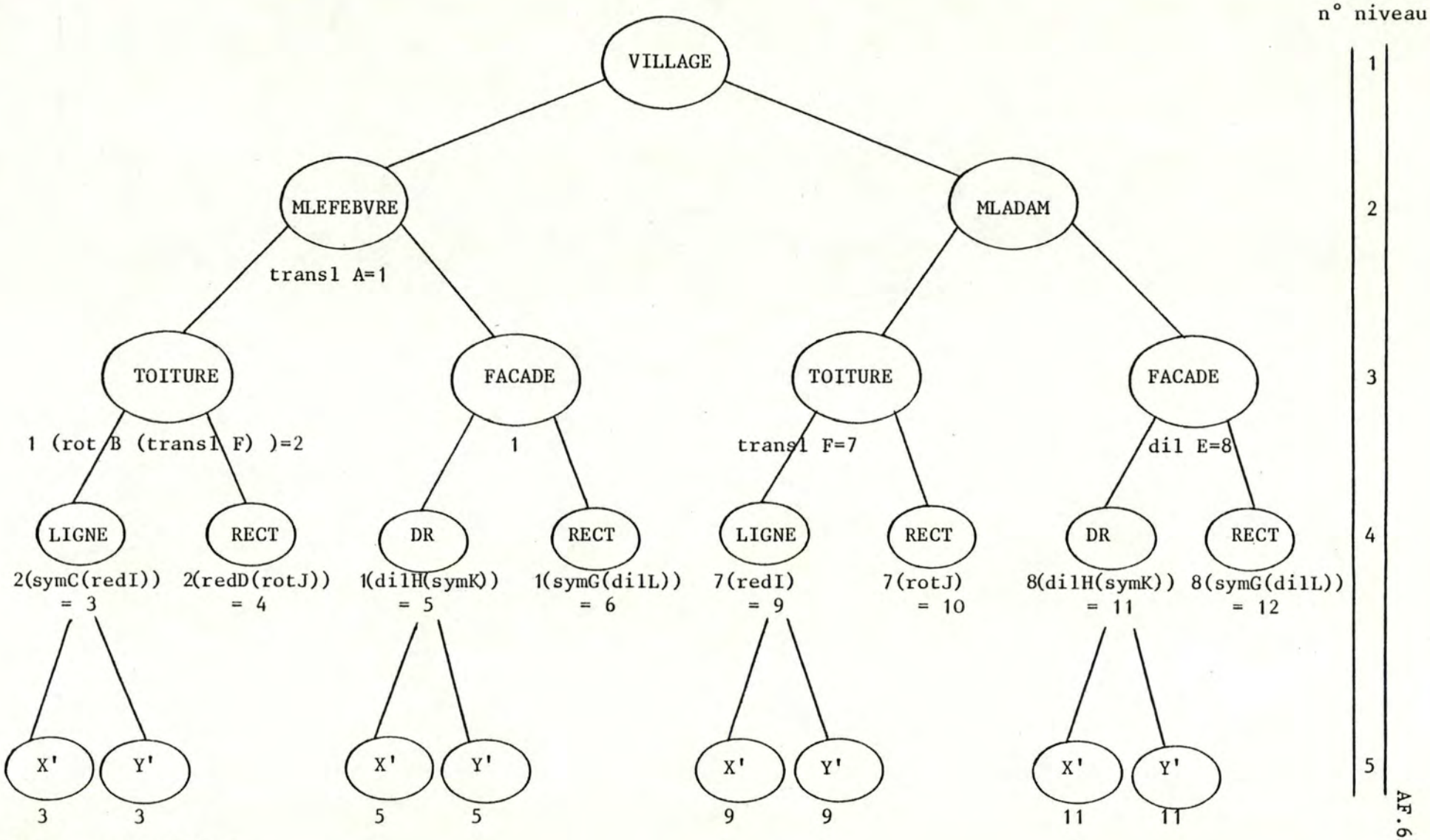
[toit : rotation B, symétrie C] ,

le produit de composition vaudra alors :

translation A (rotation B (symétrie C (translation F)))

Exemple (en notation simplifiée)

	Modificateurs globaux	Modificateurs locaux
VILLAGE (——)		
Mlefebvre : MAISON (——)	transl A	toiture : rot B ligne-toiture : sym C rect-toiture : réd D
Mladam : MAISON (——)	———	façade : dil E
MAISON ( )		
toiture : TOIT (——)	transl F	———
façade : MUR (——)	———	rect : sym G ligne : dil H
TOIT ( )		
ligne : LIGNE (——)	réd I	———
rect : RECTANGLE (——)	rot J	———
MUR (——)		
dr : LIGNE (——)	sym K	———
rect : RECTANGLE (——)	dil L	———
LIGNE ( )		
x' : DROITE (——)	———	———
y' : DROITE (——)	———	———
DROITE (——) RECTANGLE (——)	opérateurs de base	



Structure de village

avec à côté de chaque constituant, le produit de composition correspondant

LES OPERATIONS NON TRANSFERABLES

1. Comment déterminer parmi les occurences d'une opération non transférable, celle qui est la plus prioritaire ?

Il existe 2 opérations non transférables :

- couleur
- trait

Chacune de ces opérations non transférables se retrouve dans éléments (cfr. point 3.3.1.2.4. c)

- couleur E_1 E_2 E_3 E_4
- trait E_1 E_2 E_3 E_4

Pour une opération non transférable particulière, l'occurence la plus prioritaire sera celle contenue dans l'élément le plus prioritaire suivant la règle de priorité.

Comme pour les opérations non transférables, il faut traiter séparément chaque opération ; dans la règle de priorité, tous les éléments contiendront la même opération.
Ce qui ne veut pas dire qu'au niveau réalisation (algorithme), on ne pourra pas tout traiter (les ≠ opérations non transférables et les transférables) en même temps. Ici nous avons analysé cas par cas.

Introduisons la règle de priorité.
De manière simplifiée, le principe est le suivant :

pour un noeud i de la structure

les modificateurs locaux le concernant (si ∃) ont priorité sur tout ce qui concerne le niv i-1 (niv Ã si i = 1) qui lui-même a priorité sur les modificateurs globaux le concernant (si ∃).
On tiendra seulement compte en dernier lieu des instructions d'état s'il en existe.
Sinon, on prendra la valeur par défaut (V.P.D.) qui est attribuée au démarrage du système.

2. Règle de priorité entre éléments (pour une même opération NT).

Soit La structure globale d'un opérateur à n niveaux

V.P.D. valeur par défaut attribuée automatiquement si on se trouve dans le cas "couleur" et "trait". Sinon n' $\nexists$.

X_i un noeud de niveau i ($1 \leq i \leq n$)

X_j un noeud du chemin liant la racine (R) et X_i avec $1 \leq j \leq i - 1$ ($X_j \nexists$ si i = 1)

$\mathcal{P}$ le sigle "priorité sur"

$L_j = L_{ji} \mathcal{P} L_{ji-1} \mathcal{P} \ldots \mathcal{P} L_{jj+1}$

L_j = ensemble de modificateurs locaux définis dans X_j et portant sur les noeuds du chemin liant R à X_i (j+1->i) ($L_j \nexists$ si i=1)

L_{jk} avec $j+1 \leq k \leq i$ = le modificateur local défini dans X_j et portant sur X_k (le noeud de niveau k) appartenant au chemin liant R et X_i (L_{jk} peut ne pas exister)

Si dans L_{jk}, on trouve plusieurs occurences de cette opération non transférable, on donne priorité à la dernière.

G_j = modificateur global défini dans X_j et portant sur X_j (G_j peut ne pas exister et $\nexists$ si i = 1)

Si plusieurs G_j apparaissent dans la liste des modificateurs globaux définie dans X_j, alors on donne priorité au dernier.

$E_j = E_{jm} \mathcal{P} E_{jm-1} \mathcal{P} \ldots \mathcal{P} E_{j1}$ avec m non limité $0 \leq m \leq ?$

E_j = ensemble des états définis (dans le corps de X_{j-1}), avant l'appel à l'opérateur, constituant le noeud suivant du chemin liant R à X_i ($E_j \nexists$ si i=1 ou j=1)

E_{jp} = un état de cet ensemble se trouvant à la pème place dans le corps de X_{j-1} avec $1 \leq p \leq m$

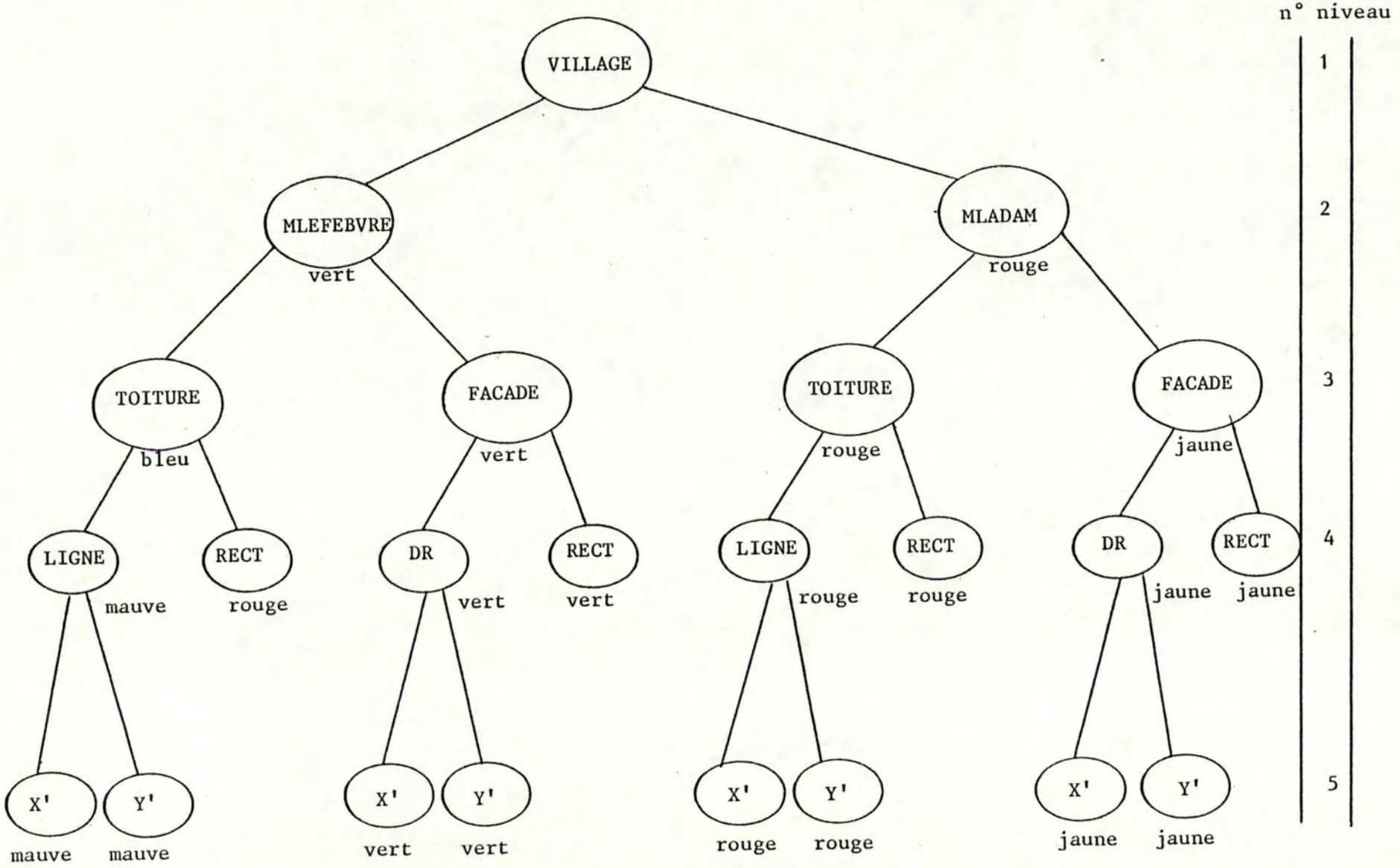
G_i = modificateur global défini dans X_i et portant sur X_i :

Alors pour le noeud X_i, on a

$(L_1 \mathcal{P} G_1) \mathcal{P} \ldots \mathcal{P} (L_j \mathcal{P} G_j \mathcal{P} E_j) \mathcal{P} \ldots \mathcal{P} G_i \mathcal{P}$ VDP

Exemple (en notation simplifiée)

	Modificateurs globaux	Modificateurs locaux
VILLAGE (——)		
couleur : rouge		
Mlefebvre : MAISON (——)	couleur verte	toiture : couleur bleu ligne-toiture : couleur mauve rect-toiture : couleur rouge façade : couleur jaune
Mladam : MAISON (——)	———	
MAISON (——)		
toiture : TOIT (——)	couleur blanc	———
façade : MUR (——)	———	rect : couleur A ligne : couleur vert
TOIT (——)		
ligne : LIGNE (——)	couleur rouge	———
rect : RECTANGLE (——)	couleur X	———
MUR (——)		
dr : LIGNE (——)	couleur Y	———
rect : RECTANGLE (——)	couleur Z	———
LIGNE (——)		
x' : DROITE (——)	———	———
y' : DROITE (——)	———	———
DROITE (——)	opérateurs de base	
RECTANGLE (——)		



Structure de village : avec à côté de chaque constituant la bonne couleur.

Développons la règle pour le noeud "X'" pour lequel, la bonne couleur est mauve.

. i = 5

  j = 1, 2, 3, 4

. L_1, L_3, L_4 $\not\exists$

  $L_2 = L_{25} \not\wp L_{24} \not\wp L_{23} \not\wp L_{22}$

  avec . L_{25}, L_{22} $\not\exists$

  . L_{24} = mauve

  . L_{23} = bleu

. G_1 $\not\exists$, G_2 = vert, G_3 = blanc, G_4 = rouge

. E_2 = rouge, E_3 $\not\exists$, E_4 $\not\exists$

. G_5 $\not\exists$

. D'après la règle, on a

$L_1 \not\wp G_1 \not\wp L_2 \not\wp G_2 \not\wp E_2 \not\wp L_3 \not\wp G_3 \not\wp E_3 \not\wp L_4 \not\wp G_4 \not\wp E_4 \not\wp G_5 \not\wp$ VDP

Ce qui donne, avec les renseignements précédents

$L_2 \not\wp G_2 \not\wp E_2 \not\wp G_3 \not\wp G_4 \not\wp$ VDP

On obtient bien la valeur désirée, mauve.

Un autre exemple pour le noeud "Toiture", pour lequel, la bonne valeur est bleu.

. i = 3

  j = 1, 2

. L_1 $\not\exists$, $L_2 = L_{23}$ = bleu

. G_1 $\not\exists$, G_2 = vert

. E_2 = rouge

. G_3 = blanc

. D'après la régle, on a

$L_1 \not\wp G_1 \not\wp L_2 \not\wp G_2 \not\wp E_2 \not\wp G_3$

Ce qui donne, avec les renseignements précédents

$L_2 \not\wp G_2 \not\wp E_3 \not\wp G_3$

On obtient bien la valeur désirée, bleu.

G) L'instruction état : ECRAN (cas particulier)

Pour une structure globale déterminée, la valeur de ECRAN retenue dépendra uniquement des renseignements ECRAN contenus dans le corps de l'opérateur composé, lié à cette structure.

Donc implicitement, cela veut dire que l'on ne tiendra pas compte des instructions état (ECRAN) définies dans le corps des opérateurs composants. C'est donc pour cette raison que nous l'avons traitée de façon particulière.

Comment déterminer la bonne valeur de ECRAN pour cette structure ?

- si le corps de l'opérateur composé ne contient pas d'instruction état ECRAN, alors une valeur par défaut sera attribuée à ECRAN.
- Sinon, c'est la dernière instruction état ECRAN définie avant la 1ère instruction opérateur, qui donnera la valeur à ECRAN.

  Nous avons fait ce choix, afin de toujours connaître valeur définitive de ECRAN avant d'exécuter le 1er opérateur.

Règle

Si VPD = valeur par défaut

E_i = ième instruction état écran du corps de l'opérateur composé, définie avant la 1ère instruction opérateur.

$(1 \leqslant i < n)$ avec E_i pouvant ne pas exister

$\mathcal{P}$ sigle "priorité sur"

Alors

$$E_n \mathcal{P} E_{n-1} \mathcal{P} \ldots \mathcal{P} E_1 \mathcal{P} VPD$$

3.3.2. LE LANGAGE L.G.

Après avoir introduit la notion d'opérateur (3.3.1.), nous allons maintenant décrire le langage L.G. (Langage Graphique), qui sera celui dans lequel on décrira un opérateur.
Rappelons que le point de départ de notre approche est l'observation des dessins, le regroupement de ceux-ci en famille et le fait que l'on ne désire pas que l'utilisateur voit un dessin comme un ensemble de traits successifs.

Nous avons de ce fait considéré que la description des dessins en terme d'autres dessins (combinaison de dessins) était la meilleure méthode de conception.
Pour ce faire, nous fournirons à l'utilisateur un ensemble de dessin de base qu'il pourra enrichir grâce aux nouveaux dessins qu'il créera.

Grâce au point 3.3.1., nous savons qu'une famille de dessins est représentée par un opérateur et que nous avons choisi de décrire cet opérateur à l'aide d'un langage particulier. Suite à l'approche qui est la nôtre, le langage permettant de décrire ces concepts sera donc fondamentalement déclaratif (descriptif).
Cependant, nous avons décidé en fonction du type d'utilisateur auquel on s'adresse, de lui fournir quelques facilités telles que : les instructions d'état qui lui permettront de modifier les valeurs retenues d'office pour les états (trait, couleur, écran) au début de l'exécution d'un opérateur et des instructions d'affectation lui permettant de calculer de manière plus aisée certaines valeurs telles que par ex : des coordonnées d'un point, un angle.

Par conséquent, le langage ne sera plus tout à fait descriptif, car l'ordre d'apparition des instructions d'état et d'affectation est important. C'est la raison pour laquelle, les mots "instruction" et "action" apparaîtront dans le langage.

Nous allons maintenant décrire le langage L.G. plus en profondeur.

Nous developperons les points suivants :

Notation	Instruction opérateur
Vocabulaire	Les paramètres
Identificateur	Les modificateurs
Nombre	Instruction d'état
Expression	Bloc
Instruction d'affectation	Opérateur

INTRODUCTION

Un programme en langage L.G. est en fait un opérateur.
Il est constitué d'actions qui sont décrites par ce que nous appellerons des "instructions".

Un opérateur se compose d'un "en-tête" et d'un corps appelé aussi "bloc". L'en-tête permet de donner un nom à l'opérateur et d'indiquer la liste de ses paramètres. Les paramètres sont des variables qui contiennent les arguments.
Le bloc est formé d'un ensemble d'instructions.
Le bloc ne peut être vide et, c'est dans ce bloc, que l'on spécifie les actions à exécuter.

NOTATION

L'aspect général d'un programme (opérateur) peut être plus précisément exprimé par des diagrammes syntaxiques.
En partant du diagramme étiquetté "opérateur", un chemin quelconque à travers les diagrammes, définit un opérateur syntaxiquement correct. Les rectangles font référence à un autre diagramme défini par ailleurs ; les symboles terminaux (ceux que l'on trouve tels quels dans un programme L.G. se trouvent dans des cercles ou dans des cases à bouts arrondis.
(Cfr. annexe C : les diagrammes syntaxiques complets du langage L.G.).

On peut aussi décrire la grammaire d'un langage avec la notation B.N.F. (Bachus-Naur-Form) traditionnelle, dans laquelle les formes syntaxiques sont exprimées par des identificateurs entre < et > . Ces identificateurs rappellent la nature ou la signification de la forme syntaxique.
On trouvera, en annexe B, la description complète de la grammaire du langage en B.N.F.

Dans la B.N.F., une éventuelle répétition d'une forme (0 à n fois) est indiquée en la mettant entre accolades { et } .

{ , } sont comme | et ::= des méta symboles.
Ils n'apparaîtront donc pas dans le texte d'un opérateur écrit en L.G., lorsqu'ils ont cette signification dans la B.N.F.

Les mots réservés ou mots clés sont le plus souvent soulignés dans les programmes écrits à la main afin de mettre en relief leur rôle de symbole à signification prédéfinie.
Le programmeur ne peut en aucun cas les utiliser pour désigner autre chose que ce pour quoi ils ont été définis, et en particulier ils ne peuvent pas être utilisés comme identificateurs.
Ils sont formés d'une suite de caractères, sans qu'aucun symbole particulier ne les distingue.
On peut trouver la table des mots réservés en annexe A.

Une construction de la forme

{<n'importe quelle suite de caractères autres que "}">}

est un commentaire : un commentaire peut être inséré en tout point du texte de l'opérateur, entre 2 identificateurs, nombres ou symboles spéciaux. Il peut être retiré du texte de l'opérateur sans en altérer le sens.

Les accolades { et } ne sont utilisées nulle part ailleurs dans le langage sauf dans des descriptions de syntaxe où elles ont un rôle de méta-symboles, au même titre que | ou ::=. Sur les ordinateurs où les accolades n'existent pas, on peut les remplacer par les symboles (* et *).

Les blancs, les fins de lignes et les commentaires sont considérés comme des séparateurs. Il peut y avoir un nombre quelconque de tels séparateurs entre 2 symboles L.G. consécutifs, mais aucun séparateur ne peut se trouver au milieu d'un identificateur, d'un nombre ou d'un symbole spécial. Enfin, il doit y avoir au moins un séparateur entre 2 nombres, identificateurs ou mots réservés.

VOCABULAIRE

Le vocabulaire de base du langage est constitué d'un certain nombre de symboles : lettres, chiffres et symboles spéciaux.

< lettre > ::= A/B/C/D/E/F/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/X/Y/Z/a/ b/c/d/e/f/g/h/i/j/k/l/m/n/o/p/q/r/s/t/u/v/w/x/y/z
< chiffre > ::= 0/1/2/3/4/5/6/7/8/9
< Symboles spéciaux > ::= { / } / ; / . / , / - / [/] / : /opérateur/ + / - / * / / / :=

IDENTIFICATEUR

Les identificateurs servent à représenter des états
des variables
des opérateurs
des opérations
des qualificateurs d'opérateurs

<identificateur > ::= < lettre> { < lettre ou chiffre >}

<lettre ou chiffre > ::= < lettre> / < chiffre >

Leur association à un qualificateur est unique dans son domaine de validité, c'est-à-dire dans un même opérateur.

Bien qu'ils puissent être de longueur quelconque, il y a pour chaque compilateur une limite, un nombre de caractères significatifs (nous choisissons les 8 premiers caractères).
En d'autres termes, 2 identificateurs ne seront distincts que s'ils diffèrent dans l'un de leurs 8 premiers caractères.

exemples d'identificateurs valides

addition cube3 pi f2m a
ceciestunidentificateurtrèslongmaisvalide
ceciestunidentificateurtrèslongmaisprobablementlemêmequeleprécédent

identificateurs invalides

7ième opérateur table.2 car-4

NOMBRE

Les nombres sont des réels et pour ceux-ci, on utilise la notation décimale habituelle. Une valeur de type réel est un élément du sous-ensemble des nombres réels que l'on peut représenter sur une machine donnée.

```
⟨nombre⟩    ::=  ⟨nombre non signé⟩ / ⟨signe⟩ ⟨nombre non signé⟩
⟨nombre non signé⟩   ::=  ⟨réel non signé⟩
⟨réel non signé⟩     ::=  ⟨chiffre⟩ { ⟨chiffre⟩ }. ⟨chiffre⟩
                          { ⟨chiffre⟩ } / ⟨chiffre⟩ { ⟨chiffre⟩ }
⟨signe⟩      ::=  + / -
```

EXPRESSION

Les expressions dénotent la manière de calculer de nouvelles valeurs des variables par le moyen d'opérateurs d'expression. Elles consistent en opérateurs d'expression et opérandes.

La syntaxe des expressions spécifie les règles de précédence dans la composition de deux grandes classes d'opérateurs d'expression. Les opérateurs d'expression multiplicatifs sont les plus prioritaires.
Des séquences composées d'opérateurs d'expression de même priorité sont exécutées de gauche à droite.

```
<expression>::=<terme> /<expression>  <opérateur additif d'expression>
               < terme> /<signe>       <terme>

<terme > ::=   <facteur> /   <terme>
               <opérateur multiplicatif d'expression>    <facteur>

<facteur>  ::=    <variable> /  <nombre non signé >  /  <expression>
```

Opérateurs d'expression

Comme les 2 opérandes des opérateurs d'expression additifs et multiplicatifs sont de type réel, alors le résultat est de type réel.

Opérateurs d'expression multiplicatifs

```
<opérateurs multiplicatifs d'expression > ::= * //
```

opérateur d'expression	opération	type opérandes	type résultat
*	multiplication	réel	réel
/	division	réel	réel

Opérateurs d'expression additifs

<opérateurs d'expression additifs > ::= + / -

opérateur d'expression	opération	type opérandes	type résultat
+	addition	réel	réel
-	soustraction	réel	réel

Utilisés avec un seul opérande, - dénote l'opposé et + l'identité.

INSTRUCTION D'AFFECTATION

Cette instruction spécifie qu'une valeur calculée doit être affectée à une variable.

<instruction d'affectation > ::= <variable > := <expression>
<variable > ::= <identificateur >

où le signe := est l'opérateur d'affectation.

INSTRUCTION OPERATEUR

Une instruction opérateur à laquelle est associé un nom < qualificateur d'opérateur > décrit la manière dont on s'est servi d'un opérateur (<identificateur d'opérateur> <liste de paramètres effectifs>) ainsi que les opérations que l'on appliquera au résultat de cette utilisation et également les opérations que l'on veut appliquer à un constituant du sous-arbre ayant pour racine le noeud associé à cette instruction opérateur.

<instruction opérateur > ::= <qualificateur d'opérateur> :
<identificateur d'opérateur>
<liste de paramètres effectifs>
<modificateur>

<liste de paramètres effectifs> ::= (<paramètres effectifs>
{,<paramètres effectifs>}) / <vide>

a) Les paramètres

Suivant sa propre définition, l'opérateur désigné possède une liste de paramètres formels qui sera vide ou non.

Dans le cas où cette liste est non vide, l'instruction opérateur possédera une liste de paramètres effectifs non vide. Chaque paramètre effectif sera substitué au paramètre formel correspondant. La correspondance sera établie par les positions respectives des paramètres dans la liste des paramètres formels et effectifs.

Dans le cas où cette liste est vide, l'instruction opérateur possédera une liste de paramètres effectifs vide.

Lors de l'exécution, le passage se fera toujours par valeur.
Il y a 3 types de paramètres effectifs : paramètres effectifs

- valeur
- variable
- expression

```
<paramètre effectif > ::= <paramètre effectif valeur> /
                          <paramètre effectif variable> /
                          <paramètre effectif expression>

<paramètre effectif valeur > ::=     <nombre>

<paramètre effectif variable > ::= <variable>

<paramètre effectif expression > ::= <expression>
```

Dans le cas d'un paramètre effectif valeur, le paramètre effectif doit être un nombre. Le paramètre formel correspondant de l'opérateur désigné, est une variable locale de celui-ci et la valeur courant du nombre est affectée à cette variable au moment de son appel.

Dans le cas d'un paramètre effectif variable, le paramètre effectif doit être une variable, et le paramètre formel correspondant de l'opérateur désigné, prend la valeur de cette variable au moment de son appel.

Dans le cas d'un paramètre effectif expression, le paramètre effectif doit être une expression. Le paramètre formel correspondant de l'opérateur désigné prend la valeur de cette expression au moment de son appel.

b) Les modificateurs

Un modificateur est constitué d'une liste de modificateurs globaux et d'une liste de modificateurs locaux, chacune de ces listes pouvant être vide.

```
< modificateur > ::= <liste de modificateurs globaux>
                     <liste de modificateurs locaux >

< liste de modificateurs globaux> ::= [<modificateur global>
                                      {, <modificateur global>}] / <vide>

< liste de modificateurs locaux > ::= [<modificateur local>
                                      {, <modificateur local>}] / < vide>
```

Un modificateur est en fait un ensemble d'opérations (dont la liste se trouve en annexe E).

Modificateur global

Le modificateur global est une opération qui porte sur l'utilisation de l'opérateur appelé par l'instruction opérateur dans laquelle le modificateur se trouve.

```
<modificateur global> ::=   <opération >

<opération>  ::=    <identificateur d'opérateur >
                    <liste de paramètres effectifs >
```

Modificateur local

Le modificateur local représente un ensemble d'opérations qui vont toutes porter sur l'utilisation de l'opérateur contenu dans l'instruction opérateur désignée par la liste des qualificateurs.

< modificateur local > ::= <liste de qualificateurs d'opérateur>
: <opération> {, <opération>}

< liste de qualificateurs d'opérateur > ::= <qualificateur d'opérateur>
{- <qualificateur d'opérateur>}

On trouvera au point 3.3.1.2.4., les priorités entre modificateurs.
Cependant, on ne peut trouver, dans une même instruction opérateur, 2 modificateurs locaux possédant la même liste de qualificateurs d'opérateur.

c) Liste de qualificateurs

<liste de qualificateurs > ::=<qualificateur d'opérateur>
{- <qualificateur d'opérateur>}

Cette liste désigne une instruction opérateur & au bloc contenant l'instruction opérateur dans laquelle la liste se trouve.

Condition :

A l'instruction contenant la liste de qualificateurs, est associé dans la structure de l'opérateur dont le bloc la contient, un noeud (1). Comme cette liste désigne une instruction opérateur, il faut comme condition de validité de la liste, que le noeud associé à l'instruction opérateur désignée appartienne au sous-arbre dont la racine est le noeud (1).

INSTRUCTION D'ETAT

Une instruction d'état décrit une utilisation particulière de l'état auquel est associé un identificateur.
Les 3 états possibles sont COULEUR définissant une couleur et une intensité de la plume
TRAIT définissant une épaisseur et un type (continu-pointillé) pour la plume
ECRAN définissant une couleur de fond et une intensité pour l'écran.
Pour la définition complète de ces états (cfr. annexe F).

```
<instruction d'état> ::= <identificateur d'état>
                         <liste de paramètres effectifs>

<identificateur d'état> ::= <identificateur>
```

L'instruction d'état contient une liste de paramètres effectifs qui seront substitués aux paramètres formels correspondants, définis dans l'en-tête de l'état.

La correspondance se fait par la position du paramètre dans la liste de paramètres formels et effectifs.
Lors de l'exécution le passage se fera toujours par valeur.
Il y a 3 types de paramètres effectifs : paramètres effectifs - valeur
- variable
- expression.

Dans le cas d'un paramètre effectif valeur, le paramètre effectif doit être un nombre. Le paramètre formel correspondant de l'état appelé est une variable locale de celui-ci et la valeur courante du nombre est affectée à cette variable au moment de son appel.

Dans le cas d'un paramètre effectif variable, le paramètre effectif doit être une variable et le paramètre formel correspondant de l'état appelé, prend la valeur de cette variable au moment de son appel.

Dans le cas d'un paramètre effectif expression, le paramètre effectif doit être une expression. Le paramètre formel correspondant de l'état appelé prend la valeur de cette expression au moment de son appel.

BLOC

Le bloc spécifie les actions à exécuter lors de l'activation d'un opérateur par une instruction opérateur.

```
<bloc > ::=      <instruction >      {; <instruction >}

<instruction>::=<instruction opérateur> / <instruction d'affection>
                                / <instruction d'état>
```

Un qualificateur d'opérateur associé à une instruction opérateur est unique dans le bloc de l'opérateur contenant l'instruction. Ce bloc doit, de plus, contenir au minimum une instruction opérateur.
D'autre part, il est interdit qu'une instruction opérateur décrive une utilisation de l'opérateur dont le bloc contient cette instruction.
Nous nous sommes ainsi, volontairement limité car le langage L.G. ne possédant que la structure séquentielle, il aurait été impossible de gérer la réussivité et savoir quand s'arrêter.

OPERATEUR

Un opérateur L.G. est constitué de 2 partie : un en-tête opérateur et un bloc.
L'en-tête d'opérateur spécifie l'identification de l'opérateur et les identificateurs des paramètres formels éventuels. Ces identificateurs doivent être tous différents.

<opérateur> ::= <en-tête d'opérateur> <bloc>

<en-tête d'opérateur> ::= opérateur <identificateur d'opérateur>
<liste de paramètres formels>

<liste de paramètres formels> ::= (<identificateur de paramètres formels>
{, <identificateur de paramètres formels>})
/ <vide>

Relation : paramètres formels de l'opérateur -
paramètres effectifs et variables de son bloc

Chaque - variable d'une instruction d'affection
- paramètre d'affection variable d'une
instruction opérateur
instruction état
- variable d'un paramètre effectif expression d'une
instruction opérateur
instruction état

peut correspondre à un paramètre formel et réciproquement.

3.4. L'EDITEUR GRAPHIQUE

3.4.1. INTRODUCTION

Suite à l'exposé des concepts de base et de leur représentation, nous étudierons dans cette partie l'éditeur graphique proprement dit. Nous parlerons d'abord de la notion de texte vu par l'utilisateur (le texte étant celui d'un opérateur) (3.4.2.), ensuite nous présenterons les particularités de notre éditeur graphique (3.4.3.), nous ferons une comparaison entre LOGO et notre éditeur (3.4.4.), nous présenterons alors les opérateurs de base, les opérations et les états (3.4.5.) fournis par les systèmes avant de développer les modes de notre éditeur graphique (3.4.6.).

Avant d'analyser les différents points que nous venons de citer, il nous paraît utile de rappeler que c'est en fonction de l'utilisateur que nous nous sommes préoccupés jusqu'à présent dans l'analyse fonctionnelle. Dans le même ordre d'idées, nous partirons des besoins de l'utilisateur pour le choix de notre éditeur.

Malgré la restriction exposée au point 3.3.2. définissant notre langage comme n'étant pas tout à fait descriptif, nous pouvons dire que notre langage L.G. est lié à notre idée de description d'une famille de dessins (opérateur).

Le langage n'a pas pour but de dire la manière d'obtenir la représentation graphique d'un dessin issu de la famille (DYNAMIQUE), mais plutôt de se situer au niveau descriptif (STATIQUE).
C'est pour cette raison que nous avons fait une distinction entre le langage et les commandes de style traçage, impression, affichage, ...
Notre système possédera donc un langage et un outil d'édition séparé.

L'existence d'une contrainte de temps pour notre travail, nous limite forcément dans le choix de l'outil que nous nous proposons de construire. Nous présenterons les commandes de base de l'éditeur graphique, que nous regrouperons sous forme de modes (cfr. 3.4.6.1.2.).
Bien évidemment, le choix de ces commandes peut être sujet à discussion, mais ce qui paraissait important pour nous, était de développer les aspects fondamentaux de notre éditeur graphique.

3.4.2. ETUDE DE LA NOTION DE TEXTE

Nous allons tout d'abord parler de la structure de fond et de forme d'un texte, et ensuite donner une définition d'un texte pour l'utilisateur.

Il était important de commencer à spécifier cette notion de texte car un opérateur est représenté par un texte, en langage L.G. (syntaxe et sémantique pour l'utilisateur).

3.4.2.1. STRUCTURE DE FORME - STRUCTURE DE FOND

Précisons tout d'abord, de manière générale, la notion de texte : le texte est au moins envisagé comme une suite de caractères. Cette définition minimale est toujours vraie puisque le caractère est un élément atomique d'un (ou de plusieurs) alphabet(s).

En considérant ses propriétés, nous pouvons définir davantage.
En effet, que l'utilisateur en ait conscience ou non, tout texte vérifie une structure de forme et une structure de fond.
La structure de forme (ou plus simplement la "forme") est l'agencement physique des caractères du texte. Les problèmes de mise en page (soulignement, justification, etc...) relèvent de cette structure. Elle concerne donc surtout la morphologie du texte.
La structure de fond (ou encore le "fond") est l'agencement logique des caractères du texte. Elle porte donc plus sur les aspects syntaxiques et sémantiques.

Pour l'utilisateur, l'intérêt qu'un texte ait une bonne structure de forme (c'est-à-dire ait la morphologie conforme à l'idée qu'il s'en fait) est non seulement la beauté de mise en page, mais aussi l'accentuation de la structure de fond : le renforcement peut garantir à l'utilisateur une meilleure perception de la sémantique et par là une facilité dans les éventuelles modifications.
En ce qui concerne la structure de fond d'un texte, l'intérêt pour l'utilisateur qu'elle soit bonne est évident : ce n'est que si cette structure est correcte que le texte est susceptible de signifier quelque chose.

Notons enfin, la possibilité d'indépendance des deux types de structures du point de vue de leur validité : l'une peut être correcte ou non, indépendamment de l'autre.

A certains égards, les propriétés "structure de fond" et "structure de forme" du concept "texte" sont liées.
D'une part, il existe des notions qui peuvent être difficilement rattachées à une seule des deux propriétés. Sans doute est-ce parce que de telles notions relèvent simultanément des deux structures ? Par exemple, outre sa fonction physique de découpe du texte, le "paragraphe" n'a t-il pas aussi une connotation sémantique lorsqu'il est considéré comme véhicule d'une idée homogène ?

D'autre part, quand un utilisateur communique à un éditeur de texte la définition d'une structure de fond syntaxique, ne le fait-il pas au moyen d'une définition morphologique ?
En d'autres termes, puisque la détermination de la syntaxe d'un texte est l'indication de toutes les séquences de caractères admises, une partie de la structure de fond (la syntaxe) est définie comme structure de forme.
D'ailleurs, un éditeur de texte ne peut vérifier si un texte est correct syntaxiquement qu'en analysant la suite des caractères du texte, c'est-à-dire en s'interrogeant sur l'aspect physique du texte.

Après avoir parlé de manière générale de la structure de fond et de forme d'un texte, précisons maintenant ce qu'il en est dans le cadre de notre travail.

La structure de fond du texte de l'opérateur respectera la syntaxe et la sémantique du langage L.G.

D'autre part, nous laisserons l'utilisateur libre de définir sa propre structure de forme en tenant compte du fait que ces deux structures peuvent être liées comme indiqué précédemment.

3.4.2.2. DEFINITION D'UN TEXTE POUR L'UTILISATEUR

Comme nous savons qu'un texte est une suite de caractères, il serait très lourd de se baser sur cette seule définition pour offrir à l'utilisateur des méthodes lui permettant d'accéder à des parties quelconques du texte. Plus précisément, il est des cas où il lui serait fastidieux de définir en termes de caractères les portions de texte sur lesquelles il veut agir.

Cela ne signifie naturellement pas que l'accès au caractère lui est inutile, mais simplement qu'il doit aussi pouvoir spécifier des parties de texte moins atomiques, c'est-à-dire des agrégations de caractères tels que le mot, le paragraphe, la phrase, la ligne.

C'est ce dernier regroupement que nous allons prendre en considération dans notre approche.
Il nous reste à présent à définir ce que nous entendons par ligne et, par extension, suite de lignes.

Désignons par ⟨retour⟩ le caractère de fin de ligne.
Nous proposons alors la définition suivante d'une ligne :

⟨ligne⟩ ::= ⟨string⟩ ⟨retour⟩ avec comme longueur maximale d'une ligne : 256 caractères (longueur d'un bloc physique sur micro)

⟨string⟩ ::= ⟨vide⟩ / ⟨caractère⟩/⟨string⟩⟨caractère⟩

⟨vide⟩représente l'ensemble vide

⟨caractère⟩ ::= Désigne tout caractère admis dans le texte, à l'exception du caractère de fin de ligne.

⟨un caractère admis dans le texte⟩ ::= ⟨lettre⟩ / ⟨chiffre⟩/⟨symbole spécial⟩ ⟨blanc⟩/⟨autre⟩

⟨lettre⟩ ::= A/B/C/D/E/F/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/X/Y/Z/ a/b/c/d/e/f/g/h/i/j/k/l/m/n/o/p/q/r/s/t/u/v/w/x/y/z

⟨chiffre⟩ ::= 0/1/2/3/4/5/6/7/8/9

⟨symbole Spécial⟩ ::= { / } / ; / . / , / - / [/] / : / = / + / - / / * / —

⟨blanc⟩ ::= le caractère blanc (espace)

⟨autre⟩ désigne tout autre caractère que l'on peut trouver sur un clavier

la suite de lignes se définit de la manière suivante :
⟨suite de lignes⟩ ::= ⟨vide⟩/ ⟨ligne⟩/⟨suite de ligne⟩ ⟨ligne⟩

3.4.3. PARTICULARITES DE L'EDITEUR GRAPHIQUE CHOISI

Après avoir présenté et expliqué les différents concepts de base ainsi que leur représentation, nous allons définir les différentes particularités de notre éditeur graphique.

Rappelons la définition d'un éditeur graphique donnée au point 2.3.4.1. :
"Un éditeur graphique est un programme qui aide un utilisateur à créer et/ou transformer un dessin de manière interactive".
Selon les concepts développés au point 3.2., les dessins sont regroupés en famille.
Les familles étant représentées par un opérateur écrit en langage L.G. (si opérateur composé) (cfr. 3.3.), la notion d'opérateur sera la notion centrale de notre éditeur graphique.

On obtiendra alors un dessin, en exécutant un opérateur pour lequel on a donné une valeur à chacun de ses paramètres formels (s'il en $\exists$). Comme indiqué précédemment, il y aura correspondance entre la structure du dessin et la structure de l'opérateur duquel il est issu (et représentant la famille). Les noms des noeuds des 2 structures sont correspondants (= le qualificateur) excepté pour la racine où on retrouvera le nom du dessin et le nom de l'opérateur.

L'éditeur nous permettra donc de créer et de modifier des opérateurs (édition d'opérateur).
Les opérateurs ainsi édités par l'utilisateur correspondront aux opérateurs composés.
De tels opérateurs composés sont construits à l'aide d'opérateurs selon la règle énoncée au point 3.3.1.1.
Ces opérateurs composés seront représentés par un texte, en langage L.G.

Dans le but d'aider l'utilisateur dans son processus d'édition, notre éditeur graphique fournira 2 manières d'éditer ce texte.

a) l'utilisateur éditera le texte d'un opérateur composé à l'aide d'un certain nombre de commandes spécifiques (mini éditeur de texte). Il écrira ou modifiera donc lui-même le texte en langage L.G.

b) notre éditeur graphique permettra de créer un nouvel opérateur composé (famille de dessins) à partir d'un élément (dessin) de cette famille. Grâce à un support graphique, l'utilisateur créera un dessin à partir duquel le système génèrera le texte en langage L.G. correspondant à la famille d'où est issu ce dessin.
En ce qui concerne la modification, l'utilisateur choisira un dessin de la famille qu'il veut modifier et il réalisera ces modifications à l'aide d'un support graphique.
Le système traduira ces modifications faites sur le dessin en changement dans le texte de la famille correspondante.

De plus, il sera possible lors de l'édition d'un opérateur composé de passer d'une manière à l'autre.
C'est le fait de pouvoir éditer le texte d'un opérateur, en ayant soit le texte soit un appui graphique tout en pouvant passer de l'un à l'autre, qui constitue la particularité de l'éditeur que nous avons choisi de construire.

Quid des dessins ?
Outre l'utilisation d'un dessin dans la 2ème manière d'éditer comme support de l'édition, l'objectif de l'éditeur sera de fournir la représentation graphique d'un dessin. Elle sera obtenue grâce à l'exécution de l'opérateur correspondant. Cependant, dans ce cas, il ne sera pas possible de transformer un dessin, toute modification devant être réalisée au niveau de l'opérateur correspondant.

L'utilisateur pourra donc stocker des dessins qui seront "définitifs".

Enfin, on constate donc que la 2ème particularité de notre éditeur graphique est le fait que nous avons distingué les commandes de l'éditeur et le langage dans lequel est écrit un opérateur.

3.4.4. COMPARAISON AVEC LE SYSTEME LOGO

Nous consacrons un paragraphe particulier aux similitudes et différences entre notre système et LOGO car celui-ci se rapproche de notre démarche.

La principale différence se situe au niveau des commandes et du langage. Dans notre système, aucune commande ne se retrouve dans le texte de l'opérateur ($\Longleftrightarrow$ procédure LOGO). Il possède des commandes et un langage. Dans LOGO, à part les commandes d'édition du texte, toutes commandes peuvent se retrouver dans le texte d'une procédure. En fait, ces commandes font partie du langage. Nous n'avons pas choisi une telle solution, car nous ne désirions avoir qu'un langage de description d'une famille de dessins.

Les primitives de base sont différentes :

Dans notre système : - les primitives représentant des outils géométriques de base (droite, cercle, rectangle, ...)
- les primitives représentant des opérations (translation, symétrie, ...)
- les primitives représentant des états (écran, couleur, trait)

On ne déplace pas une "tortue", on trace des éléments géométriques de base successivement en spécifiant l'endroit, leurs états et quelles opérations leur attacher.

Dans Logo : c'est un ensemble de primitives permettant de tracer des traits successifs. En fait on spécifie le déplacement d'une "tortue" à l'écran. (orientation, déplacement, etc ...).

LOGO est plus puissant du fait que son langage possède des structures de contrôle plus élaborées et qu'il permet également d'en créer.
Notre système ne possède que la structure séquentielle.
Nous nous sommes ainsi limités par manque de temps.

Les 2 systèmes constituent un dictionnaire de primitives que l'utilisateur enrichit en définissant ses propres procédures. C'est ce que l'on appelle des langages extensibles car les procédures créés par l'utilisateur sont les opérateurs employé(e)s de la même façon que les primitives.

Dans LOGO une procédure est créée à partir d'un éditeur texte, tandis que dans notre système, non seulement on peut le créer à partir d'un éditeur texte, mais également avec un appui graphique.

Notre langage permet, de définir, d'effectuer une opération sur n'importe quelle partie du dessin, pour autant que cette partie corresponde à un noeud de la structure du dessin (Notion de modificateur).

Logo le permet par l'intermédiaire de paramètres, comme dans notre système.
Le problème est, qu'il faut alors changer la définition de la procédure dans ce cas.
Dans notre système, le mécanisme des modificateurs nous permet d'éviter ce problème.

3.4.5. PRESENTATION DES OPERATEURS DE BASE, DES OPERATIONS ET DES ETATS

Après avoir présenté les possibilités de notre éditeur graphique (3.4.3.), nous allons présenter premièrement les opérateurs de base, ensuite les opérations fournies à l'utilisateur et enfin les états.

3.4.5.1. LES OPERATEURS DE BASE

Suite à la définition d'un opérateur (3.3.1.1.) nous savons qu'il en existe de 2 types : - les opérateurs de base
- les opérateurs composés.

Le système va donc fournir à l'utilisateur un ensemble d'opérateurs prédéfinis qui constitueront une partie des matériaux de base de son processus d'édition. L'autre partie étant constituée par les opérations et les états.
Contrairement aux opérateurs composés, ils ne sont pas écrits en langage L.G.
L'utilisateur ne pourra que les utiliser via leurs paramètres formels (si $\exists$). Il ne pourra ni les supprimer, ni les modifier dans leur définition.
Une définition précise de chaque opérateur de base disponible se trouve en annexe D.

3.4.5.2. LES OPERATIONS

Le système fournit à l'utilisateur un ensemble d'opérations prédéfinies qui constitueront une partie des matériaux de base de son processus d'édition.

L'utilisateur ne pourra que les utiliser sans pouvoir les supprimer, ni les modifier dans leur définition.
Une définition précise de chaque opération se trouve en annexe E.

3.4.5.3. LES ETATS

Le système fournit également à l'utilisateur un ensemble d'états prédéfinis qui constituera une partie des matériaux de base de son processus d'édition.

Pourquoi avoir défini des états ?
Si on désire qu'une opération n'agissant pas sur la géométrie du dessin (valeur des coordonnées du dessin), ce qui est le cas des deux opérations "couleur" et "trait", soit appliquée à plusieurs composants, on devra définir pour ceux-ci, un modificateur global contenant cette opération.
Afin d'éviter cela, on a défini ce que l'on a appelé des "états". Ils permettront donc de spécifier une opération pour plusieurs composants.
En plus de ces 2 états "couleur" et "trait", nous avons défini un 3ème état : l'état "écran", celui-ci permettant de spécifier la couleur et l'intensité de l'écran.

L'utilisateur ne pourra que les utiliser. Il ne pourra ni les supprimer, ni les modifier dans leur définition.
Lors de l'exécution d'un opérateur, une valeur sera retenue d'office pour ces 3 états (valeur définie lors du lancement du système), valeur qui sera valable tant que l'utilisateur ne la modifie pas via une instruction d'état.

Une définition précise de chaque état se trouve en annexe F.

3.4.6. LES MODES DE L'EDITEUR GRAPHIQUE

Après avoir présenté les particularités de notre éditeur graphique, les opérateurs de base, les opérations et les états fournis par le système, nous allons parler des commandes de l'éditeur graphique qui, entre autre, manipulent ces éléments.
Dans un premier temps, nous allons parler de la notion de commande (3.4.6.1.) et ensuite, nous développerons les différents modes de l'éditeur graphique (3.4.6.2.).

3.4.6.1. NOTION DE COMMANDE (DL81)

Intéressons nous d'abord aux actions que l'utilisateur effectue lors d'un processus d'édition graphique.

3.4.6.1.1. DEFINITION

Pour exprimer des actions, l'utilisateur dispose d'un ensemble de commandes. Nous définirons une commande comme un ordre donné par un utilisateur à un éditeur pendant un processus d'édition, qui est une suite alternée de transferts d'informations.

3.4.6.1.2. REGROUPEMENTS DE COMMANDES EN MODES

Un mode est un état du processus d'édition dans lequel un utilisateur peut employer un ensemble donné de commandes.

3.4.6.1.3. INTERET DES MODES

Quel est l'intérêt pour l'utilisateur, de spécifier des modes d'édition ?

Le premier avantage est la non-redondance des informations qu'il donne chaque fois qu'il utilise une commande d'un mode donné. En effet, lorsqu'il travaille dans un certain mode, il ne doit pas accompagner chacune de ses commandes, d'informations toujours identiques, à savoir celles inhérentes au mode courant. L'existence des modes peut rendre le processus d'édition plus clair aux yeux de l'utilisateur. Celui-ci peut être aidé par des regroupements judicieux des commandes suivant des critères qui l'intéressent.

3.4.6.2. DEVELOPPEMENT DES MODES DE L'EDITEUR GRAPHIQUE

Suite à la notion de mode, développée dans le paragraphe 3.4.6.1.2., nous répartirons l'ensemble de nos commandes en cinq modes différents :

- Mode Edition-opérateur (3.4.6.2.1.)
- Mode fichier-opérateur (3.4.6.2.2.)
- Mode fichier-dessin (3.4.6.2.3.)
- Mode Aide (3.4.6.2.4.)
- Mode Initialisation (3.4.6.2.5.).

Il n'y aura aucune interaction possible entre les différents modes.

Les différentes commandes de chaque mode seront définies dans l'annexe G. Ainsi, il sera nécessaire de se référer à cette annexe pour trouver l'explication complète de toutes les commandes qui seront citées ci-après.

Remarque : Dans l'éditeur graphique, tous les opérateurs auront des noms différents.

Avant de présenter les différents mode de notre éditeur, il importe de parler de la commande ? .

Cette commande permet à l'utilisateur d'obtenir à tout moment des informations qui lui sont nécessaires durant la suite de ses travaux.
Le sujet concerné par cette interrogation est déterminé en fonction de l'endroit où celle-ci a été posée.

ex. : - si l'utilisateur tape la ligne suivante :

TRACER ?

Il sera affiché la liste des paramètres effectifs de la commande TRACER ainsi que pour chaque paramètre les valeurs possibles (si ∃)

- si l'utilisateur, après être entré dans un mode, désire connaître les commandes disponibles dans celui-ci, il peut les obtenir grâce à la commande ?

Cette commande est donc disponible à tout moment dans toute commande des différents modes.

3.4.6.2.1. LE MODE EDITION-OPERATEUR

Le mode édition permet de créer ou de modifier le texte d'un opérateur composé. Par définition des opérateurs de base, ce mode ne les concerne pas.
Dans ce mode, nous avons choisi de ne pas distinguer explicitement la création de la modification.

Cette distinction se fera de la manière suivante :
lorsque l'utilisateur désire éditer un opérateur (il a donné le nom de l'opérateur choisi), deux situations peuvent se présenter :

- soit l'opérateur choisi n'existe pas dans le système, c'est-à-dire qu'il est édité pour la première fois.
  L'utilisateur se trouve alors implicitement dans le cas d'une création.
- soit l'opérateur choisi existe dans le système, c'est-à-dire qu'il a fait l'objet d'une édition précédente.
  L'utilisateur se trouve alors implicitement dans le cas d'une modification.

Dans les 2 cas, l'utilisateur réalisera une édition mais sans en spécifier le type. Il choisira au départ son mode d'édition : sous-mode texte
sous-mode graphique.

Il pourra de plus, en cours d'édition, passer du sous-mode texte au sous-mode graphique et inversément.
Cependant, nous pouvons remarquer qu'un problème se pose lorsque l'on effectue une suite d'éditions sur un même opérateur.
Si celui-ci intervient dans la définition d'autres opérateurs, les changements se répercuteront alors sur toutes ces utilisations antérieures.
Deux positions peuvent être adoptées vis-à-vis de ce problème :

1) - soit accepter ces répercutions
2) - soit les refuser.

Alors comment gérer chacun de ces cas ?

a) ⟶ versions multiples
b) ⟶ MAJ par l'utilisateur.

a) versions multiples

On peut associer un numéro de version à chaque texte modifié. Le numéro 1 étant associé à la version créée au départ, le dernier numéro étant le seul disponible pour les utilisations ultérieures.
Donc dans le corps de l'opérateur, à chaque nom d'opérateur contenu dans une instruction opérateur sera associé un n° de version.

Avantages : Cette méthode nous permet de modifier le texte d'un opérateur sans conséquences sur les utilisations antérieures.

Inconvénients : Cette méthode va, après quelques modifications et si le nombre d'utilisations entre les modifications est important, voir le nombre de versions proliférer de manière exhorbitante.

Pour chaque opérateur, on ne pourra utiliser que la dernière version mais on devra mémoriser toutes les versions intermédiaires.

b) MAJ par l'utilisateur

Comme dans l'hypothèse choisie, il n'y a pas de versions multiples tout appel sera relié à la version modifiée.

Il peut y avoir incohérence au niveau syntaxe (langage L.G.).

C'est seulement lorsque l'utilisateur fera appel à un opérateur contenant une ancienne version que lui sera signalée cette incohérence.

Il devra alors rectifier lui-même le problème.

Avantages : Gain de place car pas de versions multiples.

Pas de travail de recherche.

Inconvénients : Perte de cohérence rectifiée après coups.

c) Remarque

Le système offre une possibilité de contourner les problèmes découlant de certaines modifications.

L'utilisateur au lieu de modifier le texte de l'opérateur en L.G., peut créer un nouvel opérateur contenant en plus de l'appel à l'opérateur qu'il désirait modifier, les modifications elles-mêmes.

Ce principe ne pourra fonctionner que dans le cas d'ajout d'instructions opérateur.

Ainsi on évite totalement les problèmes liés aux versions qui seraient devenues antérieures.

Exemple : soit Maison (a, b) ;

x : x' (a) ;
y : y' (b) .

On désire ajouter l'instruction opérateur suivante

z : z' (2)

A la place d'avoir

Maison (a, b) ;

x : x' (a) ;
y : y' (b) ;
z : z' (2) .

On peut selon cette idée avoir

M (a, b) ;

mlef : maison (a, b) ;
z : z' (2) .

L'utilisateur peut également contourner ce problème en réalisant une copie de l'opérateur qu'il désire modifier. Il travaillera alors sur la copie qui portera un nom différent.

d) Conclusion

La 2ème possibilité (b) sera choisie car : - elle force l'utilisateur à vérifier la cohérence de son système ; cela lui demande donc d'être précis et complet dans ses travaux.

- elle ne fait pas proliférer de manière exhorbitante les versions. (rappelons que cet éditeur a pour but de fonctionner sur Microordinateur).

Nous allons maintenant développer les 2 sous-modes cités précédemment

- le sous-mode texte
- le sous-mode graphique.

Rappelons que l'utilisateur, lorsqu'il désirera réaliser une édition, choisira en même temps le sous-mode soit texte
soit graphique.

Ceci sera réalisé via les commandes EDITEXT (EDT)
EDITGRAPH (EDG)

qui seront expliquées dans l'annexe G.

3.4.6.2.1.1. Le sous-mode texte

On peut définir un opérateur par l'intermédiaire des commandes du sous-mode texte. Celles-ci permettent à l'utilisateur d'éditer c'est-à-dire de créer un nouvel opérateur (son texte) ou de modifier le contenu du texte d'un opérateur défini. Dans ce sous-mode, la notion de texte sera donc centrale. L'utilisateur disposera d'un miniéditeur de texte de type écran lui permettant de mettre à jour constamment la vision à l'écran du texte édité et lui indiquant, par positionnement du curseur, l'endroit où doivent agir les commandes. La notion de texte est prise dans le sens précisé au point 3.4.2.

Les commandes de ce mode sont réparties dans 8 classes d'actions :

- Mouvement du curseur
- Insertion et suppression
- Sortie du sous-mode texte
- Sortie de l'édition
- Vérification de syntaxe
- Défilement de l'écran
- Sauvetage du texte
- structure.

Afin d'entrer dans ce sous-mode, l'utilisateur disposera de la commande EDITEXT. Si l'édition concerne un opérateur existant, l'utilisateur disposera alors du texte de celui-ci (en langage L.G.) et pourra commencer à la modifier. Sinon, il créera à l'aide des commandes disponibles dans ce sous-mode, le texte (L.G.) décrivant le nouvel opérateur. Il disposera de 3 classes d'actions afin de traiter (créer et/ou modifier) le texte proprement dit : Mouvement du curseur, Insertion et suppression, Défilement de l'écran. Les 3 classes seront également utilisées afin de formuler les commandes lors de l'édition et gérer l'écran.
A tout moment l'utilisateur pourra faire vérifier la cohérence du texte en cours d'édition par rapport à la syntaxe du langage L.G. et ce, via la commande VS, ainsi que demander des renseignements concernant la structure d'un opérateur.
Il pourra également, à tout moment sauver le texte en cours d'édition via la commande de sauvetage du texte. Il sera possible de passer du sous-mode texte au sous-mode graphique tout en restant en édition du même opérateur et ce, via la commande GOGRAPH.
La sortie de l'édition proprement dite se fera via 2 commandes (voir classe d'action : sortie de l'édition).

3.4.6.2.1.2. Le sous-mode graphique

Nous avons vu que nous pouvons également définir un opérateur par l'intermédiaire de commandes du sous-mode graphique. Celles-ci permettent à l'utilisateur de créer ou de modifier le texte de l'opérateur choisi, avec appui graphique. Elles sont réparties dans 9 classes d'actions

- Sortie de l'édition
- Sortie du sous-mode graphique
- Dessin
- Marquage
- Curseur

- Définition de paramètres
- Ecran
- Sauvetage
- Structure.

L'utilisateur peut entrer dans le sous-mode, de 2 manières
. soit en commençant une nouvelle édition et en entrant directement en sous-mode graphique grâce à la commande EDITGRAPH.
. soit en passant du sous-mode texte en sous-mode graphique, grâce à la commande GOGRAPH.
L'édition se fera à l'aide d'un support graphique.

Dans le cas d'une création, l'utilisateur disposera d'un ensemble de commandes lui permettant de créer une représentation graphique d'un dessin à partir de laquelle le système génèrera le texte de l'opérateur correspondant.
Ces commandes sont celles des 6 classes d'actions suivantes :

- dessin
- marquage
- curseur
- écran
- définition de paramètres
- structure.

Dans le cas d'une modification, il faudra exécuter l'opérateur afin d'obtenir une représentation graphique d'un dessin de cette famille. Pour cela, il faudra que le texte de l'opérateur soit syntaxiquement correct vis-à-vis du langage L.G. De plus, à la fin de l'exécution, le système fournira une liste contenant pour chaque variable du texte de l'opérateur en cours d'édition, sa valeur finale, ainsi qu'un renseignement indiquant si cette variable apparaît parmi les paramètres formels ou non. Cette liste permettra, dans ce sous-mode, de connaître les variables du texte et leur valeur afin de pouvoir les utiliser. (les règles d'utilisation sont définies en annexe G, point I.2.5. ; définition des paramètres). Il disposera alors en modification, des mêmes commandes que dans le cas de la création.

La classe DEFINITION DE PARAMETRES, permettra à l'utilisateur de définir un ou plusieurs paramètres formels pour l'opérateur en cours d'édition ainsi que l'attribution d'une valeur à chacun de ces paramètres. Il pourra alors utiliser des paramètres effectifs variables (dont la valeur est fixée) dans les commandes de la classe d'actions "dessin".

De plus, cette classe permettra de supprimer des paramètres formels de l'en-tête de l'opérateur en cours d'édition, avec les changements correspondants dans son texte. Cependant, des restrictions sont à apporter quand à l'utilisation de cette commande (cfr. Annexe G point I.2.5.). L'utilisateur pourra également obtenir la valeur de tout paramètre contenu dans la liste des variables.

On trouvera en annexe G les répercutions des commandes de cette classe sur le texte de l'opérateur en cours d'édition, ainsi que leur définition.

La classe STRUCTURE permettra à l'utilisateur d'obtenir l'arbre de structure de l'opérateur en cours d'édition.

La classe DESSIN recouvre toutes les commandes liées à la création et la modification proprement dite du texte d'un opérateur à l'aide de la représentation graphique du dessin support de l'édition.
On retrouve 3 types de commandes dessin :

- la commande de traçage
- les commandes opérations
- les commandes états

Les types de commande vont engendrer des modifications sur le texte de l'opérateur en cours d'édition. C'est à ce niveau que sera fait le lien entre le sous-mode graphique et le texte en langage L.G.
On développera les répercutions de chaque commande de ces 3 types sur le texte en langage L.G. au moment de la définition de celles-ci dans l'annexe G.

.. La commande de traçage "tracer" va permettre d'obtenir la représentation graphique d'une réalisation particulière d'un opérateur c'est-à-dire un dessin.

. Les commandes opérations vont permettre d'appliquer une opération à un constituant quelconque de la structure du dessin support de l'édition.
. Les commandes états permettront de définir et de changer la valeur des états.

Chaque commande de ces 3 types sera exécutée directement après être tapée, avec répercution sur la représentation graphique de l'écran. On pourrait fournir à l'utilisateur une commande lui permettant de définir un ensemble de commandes qui ne seront exécutées qu'après avoir été toutes tapées.

Ainsi il pourrait définir, par exemple, 3 commandes opérations successives portant sur un même constituant, pour lesquelles l'effet global seul, sera affiché et non les effets particuliers de chaque opération.

Nous n'avons cependant pas approfondi ce point, qui pourrait faire l'objet d'une étude ultérieure.

Nous allons maintenant développer les différentes manières de désigner un constituant d'une structure en sous-mode graphique. Ceci nous intéresse dans le cadre des commandes opérations.

Désigner un dessin constituant d'un dessin composé, revient à désigner un noeud particulier de la structure du dessin composé à l'exception du noeud racine.
On peut désigner un dessin constituant d'un dessin composé de la manière suivante :

a) En donnant le nom du dessin constituant dans le dessin composé
b) En donnant le numéro du dessin constituant dans la structure du dessin composé
c) En positionnant une croix sur le dessin constituant
d) En encadrant le dessin constituant.

a) En donnant le nom du dessin constituant dans le dessin composé

L'utilisateur désignera un dessin constituant par une liste de qualificateurs. Pour obtenir cette liste, il aura la possibilité de visualiser la structure du dessin composé. Il construira cette liste en retenant le nom de chacun des noeuds successifs du chemin allant de la racine de cette structure au noeud correspondant au dessin constituant.

On choisit que l'utilisateur ne soit pas obligé d'indiquer systématiquement le nom de la racine de la structure, par facilité.

b) En donnant le numéro du dessin constituant dans la structure du dessin composé

Dans la structure du dessin composé, en plus du qualificateur associé à chaque noeud, on peut envisager d'indiquer un numéro qui serait composé du numéro de niveau du noeud dans la structure et sa place dans le niveau.

Ex.

ce noeud aurait le numéro 2.2

Comme à chaque dessin constituant correspond un noeud dans la structure, il suffit d'afficher celle-ci afin d'obtenir le numéro recherché.

c) En positionnant une croix sur le dessin constituant

L'utilisateur désignera sur l'écran à l'aide d'une croix le dessin constituant désiré.

d) En encadrant le dessin

L'utilisateur encadrera d'un rectangle une partie d'écran dans laquelle se trouve le dessin constituant.

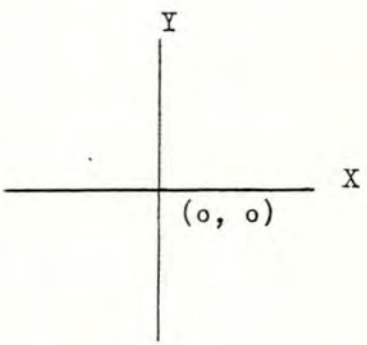
Problèmes : que faire si une partie ou un ou plusieurs autres dessins composants se trouvent dans le rectangle ? Une solution serait de demander à l'utilisateur de préciser l'encadrement, de modifier celui-ci.

Afin de faciliter la détermination de certains paramètres des commandes de la classe "dessin", nous avons introduit la notion de marque ainsi qu'une série de commandes liées aux MARQUES.
Nous allons d'abord définir la notion de marque.
La notion de marque est fournie à l'utilisateur pour faciliter la désignation (et la mémorisation) d'un endroit précis de l'écran.
A la place de donner des coordonnées traditionnelles, il pourra travailler avec les marques.
Il existera 2 types de marques - déterminées
- non déterminées.

- une marque sera déterminée lorsqu'elle possède un nom
- dans le cas contraire, elle est dite non déterminée.

Les commandes liées aux marques permettront de définir, de visualiser, d'effacer et de supprimer une marque.

De plus, un certain nombre de commandes seront liées à l'ECRAN.
Physiquement la surface d'un écran est limitée. Il paraît peu concevable de limiter un dessin à cet écran. Un dessin possédera donc un espace propre (espace dessin).
L'espace sera celui défini par le système d'axe suivant :



Cet espace n'aura pas de limite au niveau des coordonnées.
Cet espace sera le même pour tous les dessins, de ce fait, toutes les coordonnées utilisées dans la définition d'un opérateur seront définies en fonction de cet espace.
Ce raisonnement est conceptuel, mais au niveau réalisation, il sera nécessaire de limiter l'espace dessin.

L'écran dont la surface ne peut être changée, constitue une fenêtre sur cet espace. Cette fenêtre pourra se déplacer dans cet espace. Conceptuellement, cette fenêtre pourra changer de dimension, permettant de visualiser - une plus grande partie de l'espace (ce qui revient à une diminution du dessin)

- une plus petite partie de l'espace (ce qui revient à un agrandissement du dessin).

Au départ, le coin inférieur gauche de l'écran correspondra à la coordonnée (o, o) de l'espace dessin.
A tout moment, l'utilisateur disposera d'une information lui permettant de situer l'écran dans l'espace dessin.
De ce fait, nous pouvons dire que les coordonnées utilisées dans le texte d'un opérateur seront exprimées dans l'espace dessin.

Nous aurons également des commandes de déplacement du CURSEUR à l'écran.

L'utilisateur pourra aussi, à tout moment, sauver le texte de l'opérateur en cours d'édition, texte généré par le système, et ceci via la commande de SAUVETAGE.

Il sera possible de passer du sous-mode graphique au sous-mode texte tout en restant en édition du même opérateur et ce via la commande "gotext" (SORTIE DU SOUS-MODE GRAPHIQUE).

Enfin, la SORTIE DE L'EDITION proprement dite se fera via 2 commandes (voir classe d'action : sortie de l'édition).

3.4.6.2.2. LE MODE FICHIER-OPERATEUR

Le mode fichier-opérateur permet d'afficher à l'écran le texte d'un opérateur.
De plus, il sera possible de vérifier la cohérence de ce texte par rapport au langage L.G. et ce via la commande V.S.
Il sera possible d'imprimer ce texte, en faire une copie qui aura un nom différent, changer son nom et supprimer cet opérateur du système.

3.4.6.2.3. LE MODE FICHIER-DESSIN

Le mode fichier-dessin permet d'obtenir une représentation graphique d'un dessin sur écran ou imprimante.
Ces représentations seront obtenues en exécutant l'opérateur définissant la famille à laquelle appartient le dessin dont on veut obtenir la représentation graphique.

Il sera possible de mémoriser ces représentations graphiques en leur donnant un nom (tous les noms devront être différents), de changer ce nom, de les afficher par la suite, de les imprimer, de les supprimer du système.

3.4.6.2.4. LE MODE AIDE

A tout moment, l'utilisateur aura la possibilité d'obtenir des informations sur l'état du système, grâce aux commandes du mode aide. Il sera informé sur les opérateurs, les états, les commandes et les opérations.

3.4.6.2.5. LE MODE INITIALISATION

Ce mode permet de définir ou de modifier le rapport en l'axe des X et des Y, rapport qui sera utilisé lors de l'affichage à l'écran ou l'impression de la représentation graphique d'un dessin. Dans ces cas, ce sera toujours la valeur courante de ce rapport qui sera utilisée. D'aucune manière, ce rapport ne peut être particulier à une famille de dessins.

PARTIE 4 : PRINCIPES DE REALISATION

(Lefebvre J. sauf
Ladam J.-P. : 4.4.3.4.
4.4.3.5.
4.5.2.3.3.)

Nous avons expliqué dans l'analyse fonctionnelle, que l'objectif de l'utilisateur était de créer des dessins. Nous les avons regroupés en familles, qui sont décrites par des opérateurs.

Au niveau de l'éditeur graphique, l'opérateur sera donc le noeud central. Il sera le moyen dont disposera l'utilisateur pour obtenir ses dessins.

La création d'une famille de dessins se résumera à la création d'un texte, respectant la syntaxe du langage LG. Il en va de même pour la modification.
Bien que notre éditeur dispose de 5 modes (cfr 3.4.6.), le mode le plus important et le plus complexe est le mode édition-opérateur.

L'utilisateur peut éditer un opérateur composé, de deux manières différentes, via ce que l'on a appelé le sous-mode texte et le sous-mode graphique. Chacun de ces deux sous-modes a ses caractéristiques propres et le passage d'un sous-mode à l'autre est soumis à certaines conditions (cfr : analyse fonctionnelle).

Analyser le fonctionnement du sous-mode graphique, va nous amener à aborder entre autre, le problème de l'exécution d'un opérateur, de l'affichage à l'écran de la représentation graphique du dessin support de l'édition. Nous aborderons à ce niveau, des problèmes dont certains sont similaires à ceux rencontrés dans le mode fichier-dessin.

Vu la contrainte de temps qui nous est imposée, nous nous limiterons donc à exposer, dans cette quatrième partie, les principaux problèmes inhérents à ce mode central, qu' est le mode édition opérateur.

4.1. INTRODUCTION

Le fondement principal de notre éditeur graphique est de permettre à l'utilisateur, à partir d'un opérateur décrivant une famille de dessins et représenté par un texte écrit en langage LG, d'obtenir la représentation graphique d'un dessin de cette famille et ceci via l'exécution de l'opérateur décrivant la famille.

Nous avons précisé que nous ne développerions dans cette quatrième partie, que le mode édition opérateur. Ce mode permet à l'utilisateur de créer et modifier ses propres opérateurs (opérateurs composés). Il éditera de tels opérateurs grâce à deux sous-modes dont les principes sont totalement différents.

- Le sous-mode texte permet via un éditeur de texte, d'éditer le texte d'un opérateur. L'utilisateur ne dispose alors d'aucun appui graphique.

- L'autre sous-mode, le sous-mode graphique, permet par contre de créer ou modifier le texte d'un opérateur à l'aide d'un support graphique, certaines commandes de l'utilisateur ayant des répercussions sur le texte.

- De plus, il sera possible de passer d'un sous-mode à l'autre sous certaines conditions.

On voit donc à ce niveau, l'importance qu'a le texte d'un opérateur dans ce mode.

Il est possible de schématiser le fonctionnement de ce mode de la manière suivante :

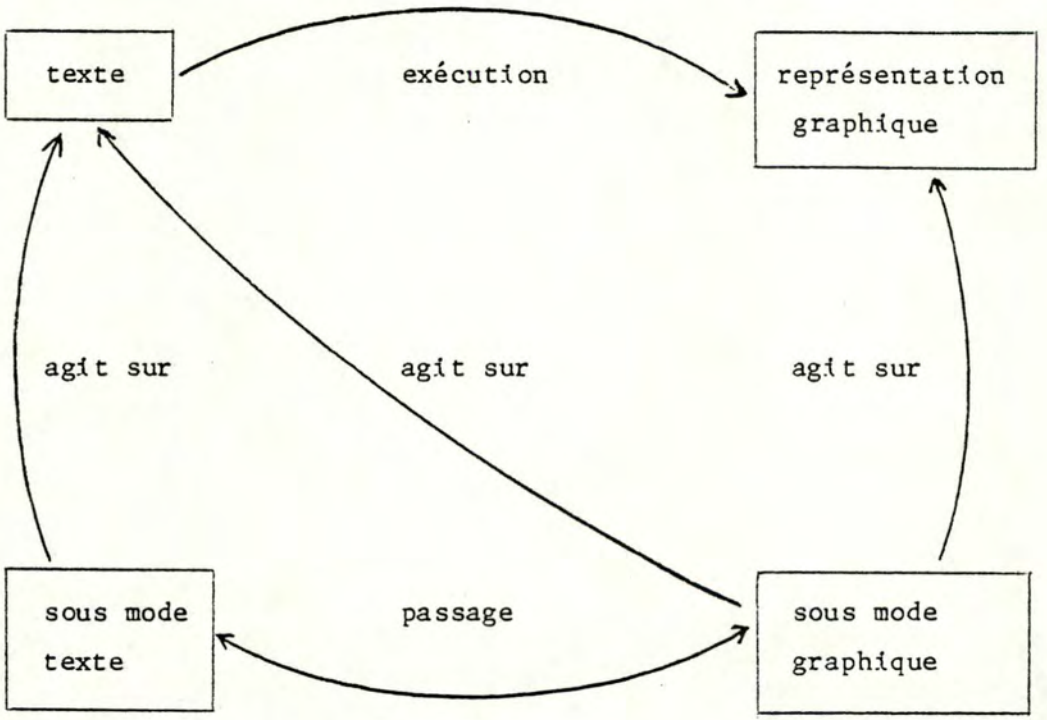


figure 1

Ce schéma constituera le fil conducteur de toute cette partie concernant les principes de réalisation.

Nous allons donc, tout d'abord parler du texte d'un opérateur composé (4.2). Puis, nous aborderons le problème de la représentation graphique d'un dessin (4.3).
Ensuite, nous parlerons de l'exécution d'un opérateur, qui va permettre de passer de l'un à l'autre (4.4), puis nous traiterons l'édition d'un opérateur (4.5).
Enfin, nous terminerons par une remarque concernant le mode fichier-dessin (4.6) et par la conclusion (4.7).

4.2. LE TEXTE D'UN OPERATEUR COMPOSE

Comme nous avons vu qu'un opérateur composé était représenté par un texte en langage LG, nous avons dans l'analyse fonctionnelle, défini le concept de texte du point de vue de l'utilisateur. Nous allons maintenant développer cette notion du point de vue de l'éditeur graphique.

Le mode le plus important étant le mode édition opérateur, ce sera donc celui-ci qui conditionnera le choix de la représentation du texte d'un opérateur composé.

4.2.1. LE TEXTE VU PAR LE SOUS-MODE TEXTE ET LE SOUS-MODE GRAPHIQUE : 2 STRUCTURES .

Dans chacun de ces deux sous-modes, le texte est vu de manière différente :

- pour le sous-mode texte :

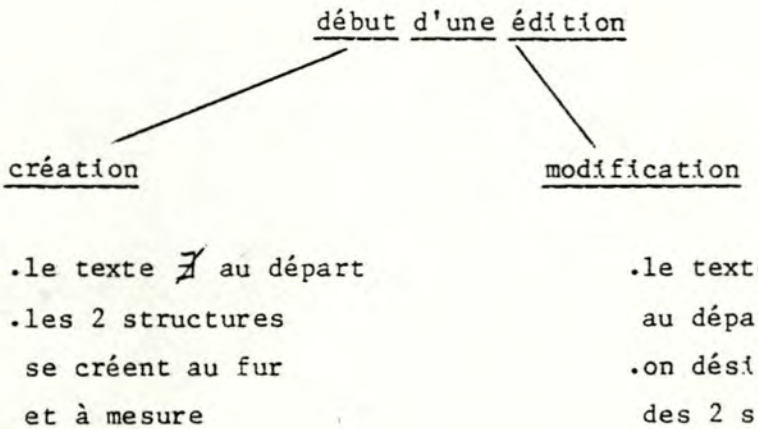
  l'utilisateur voyant le texte comme un ensemble de lignes et comme il doit pouvoir facilement accéder, ajouter, supprimer et travailler dans une ligne, on doit disposer, liée au texte, d'une structure basée sur la notion de LIGNE.

- pour le sous-mode graphique :

comme certaines commandes de
l'utilisateur vont, soit agir sur les
instructions du texte, soit être
transformées en instructions du type
du langage LG (l'en-tête de
l'opérateur étant assimilée à une
instruction), on doit disposer,
liée au texte, d'une structure
basée sur la notion d'INSTRUCTION.

Donc, au début d'une édition, compte tenu du fait que l'on peut passer d'un sous-mode à l'autre, on désire disposer de deux structures liées au texte.

On obtient donc le schéma suivant :



Cependant, disposer de ces deux structures en même temps, au début d'une édition de type modification, n'est pas toujours possible.
En effet, pouvoir disposer d'une structure basée sur la notion d'instruction, implique que la syntaxe du texte soit correcte. Or, à la sortie d'une édition en sous-mode texte, nous n'imposons aucun contrôle de syntaxe. De plus, aucune vérification automatique n'est effectuée au fur et à mesure que l'utilisateur tape le texte de l'opérateur. Il se peut donc, qu'à la sortie le texte ne soit pas correct syntaxiquement.

On peut donc dire qu'en entrant en édition du type modification, on peut toujours disposer de la structure basée sur la notion de ligne. Ce qui n'est pas toujours le cas pour l'autre structure.

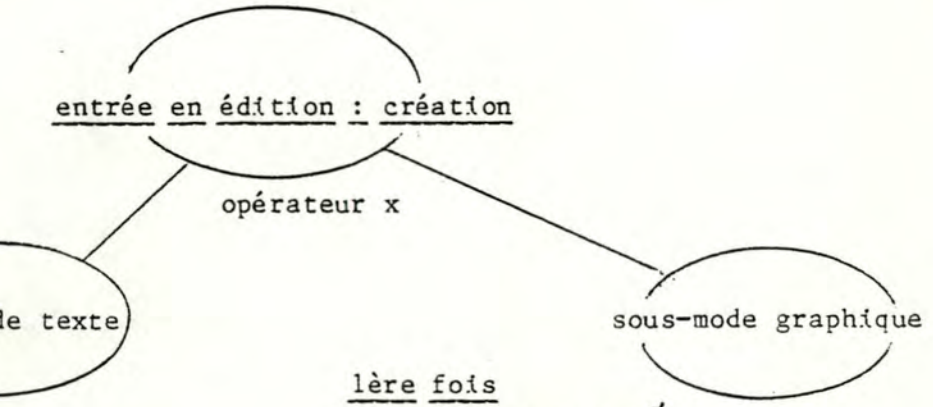
On pourrait adopter une solution consistant à vérifier la syntaxe du texte, pour voir si on peut créer ou non une telle structure. Cette solution est cependant trop coûteuse en temps.
D'autre part, comme la condition pour entrer en sous-mode graphique est que la syntaxe du texte de l'opérateur soit correcte (afin de pouvoir obtenir le support graphique), on pourrait ne créer cette structure que lors de l'entrée dans ce sous-mode. Ceci se ferait au fur et à mesure de l'exécution de l'opérateur (chose nécessaire afin d'obtenir le support graphique).

Nous adopterons cette solution car, elle nous permet de faire d'une pierre deux coups et ainsi de gagner du temps.
On construira la structure basée sur la notion d'instruction, lors de l'exécution de l'opérateur en cours d'édition (afin d'obtenir le support graphique) et ceci à chaque entrée dans ce sous-mode. Cependant, on obtiendra cette structure uniquement si l'exécution ne rencontre pas de problèmes (erreurs de syntaxe).

Pourquoi créer une nouvelle structure à chaque entrée dans ce sous-mode?

Car le fait de pouvoir passer du sous-mode graphique au sous-mode texte, peut rendre la syntaxe incorrecte. Le même problème existe lorsqu'on entre directement dans le sous-mode graphique, la syntaxe pouvant être incorrecte.

En conclusion, on obtient le schéma suivant :



lère fois

.texte de X $\nexists$
.pas de structure en lignes
.la structure en lignes se construit et est mise à jour au fur et à mesure de la création

lère fois

.texte de X $\nexists$
.pas de support graphique
.pas de structure en instructions
.la structure en en instruction se construit et est mise à jour au fur et à mesure de la création

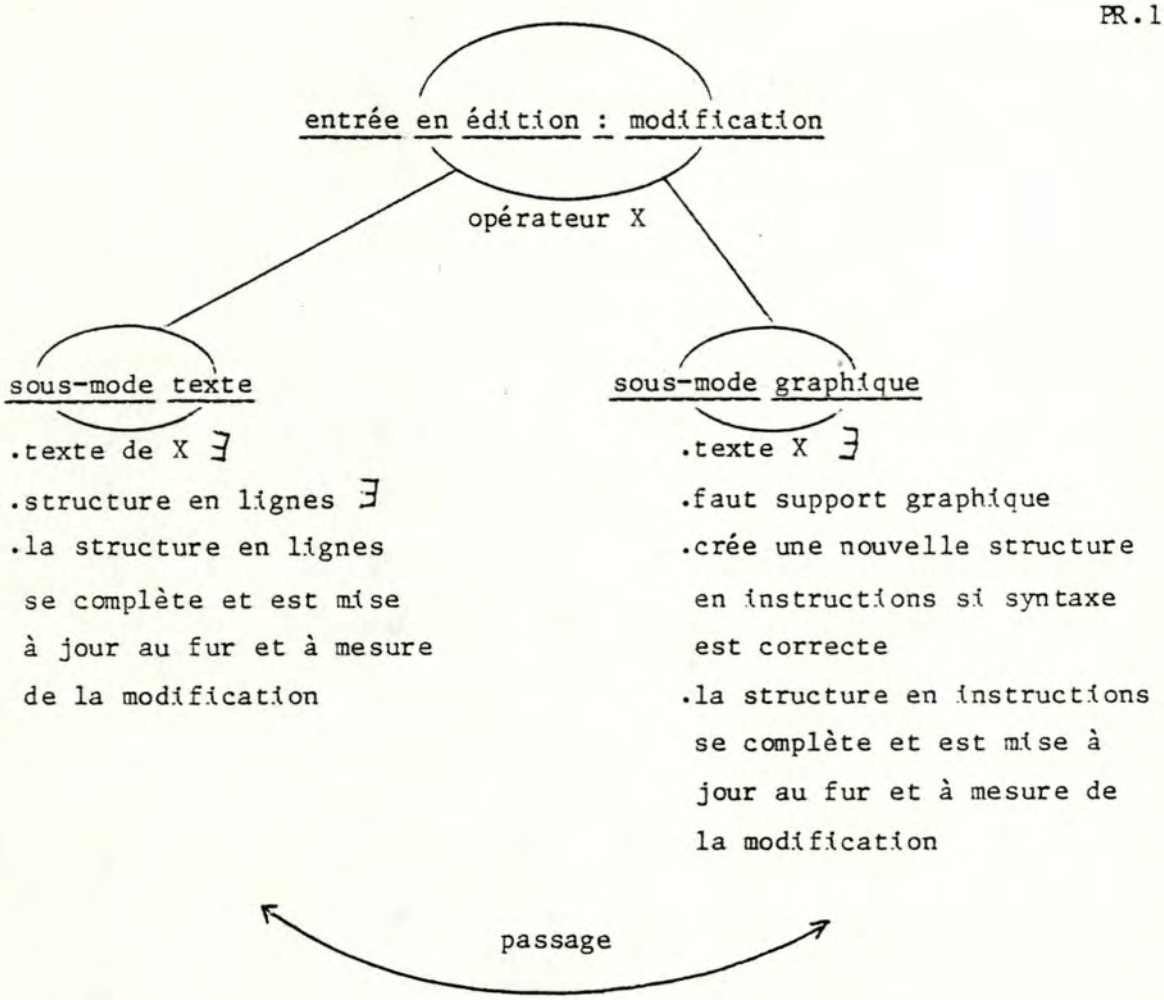
passage

autres fois

.texte de X $\exists$
.structure en lignes $\exists$
.la structure en lignes se complète et est mise a jour au fur et à mesure de la création

autres fois

.texte de X $\exists$
.faut support graphique
.crée une nouvelle structure en instructions si la syntaxe est correcte.
.la structure en instructions se complète et est mise a jour au fur et à mesure de la création.



4.2.2. REPRESENTATION DU TEXTE D'UN OPERATEUR COMPOSE EN MEMOIRE

Nous avons vu qu'il existe deux structures liées au texte, liées à chacun des sous-modes du mode édition-opérateur.

Cependant, quel choix effectuer au niveau physique pour le texte ?

Nous considérerons tout au long de cette quatrième partie que nous disposons en mémoire centrale, des textes des opérateurs nécessaires au travail en cours.

Deux solutions sont possibles au niveau représentation :

-dédoublement physique du texte

(deux versions)

-version unique du texte.

- Dédoublement physique du texte (deux versions)

pour chaque structure, nous avons la même information, mais organisée de manière différente. L'une est organisée en tenant compte de la notion de ligne et l'autre de la notion d'instruction. Une telle solution nous paraît coûteuse du point de vue

* de la place mémoire (redondance d'information) alors que nous travaillons sur microordinateur dont la taille mémoire est limitée

* du temps (mise à jour double lors d'un changement d'information).

- Version unique du texte

on a une seule version du texte ainsi qu'un mécanisme permettant d'obtenir la structure sous forme de lignes et sous forme d'instructions. Cette solution est celle que nous préconisons. Pourquoi ?

* gain de place mémoire

* facilité de mise à jour du texte (une seule version à mettre à jour) = gain de temps

Par manque de temps, et vu le fait qu'une analyse détaillée de la représentation d'un texte est un problème classique, nous nous limiterons ici, à proposer une manière d'organiser la représentation du texte.

En fonction de quoi l'organiser?

- notion de ligne

- notion d'instruction

- indépendamment de ces deux notions.

Comme expliqué précédemment, nous avons préconisé de ne disposer que d'une version unique du texte en mémoire centrale. Organiser cette version unique en fonction de la notion d'instruction est impossible. Pourquoi ? Car nous savons que la syntaxe du texte n'est pas toujours correcte, et qu'on ne peut vérifier l'exactitude de celle-ci et donc créer la structure en instruction, que lors de l'exécution faite lors de l'entrée en sous-mode graphique.

Une organisation indépendante des notions de ligne et d'instruction, est possible. Mais il serait plus judicieux d'organiser la représentation en fonction de la ligne pour les raisons suivantes:

- Facilite le travail pour le mode texte à cause du mini-éditeur de texte permettant d'insérer, d'ajouter, supprimer et modifier des lignes (besoin d'accès rapide).

- Certaines commandes du sous-mode graphique ont des répercussions sur le texte de l'opérateur en cours d'édition (cfr 3.4.6.1.2).
  En ce qui concerne l'ajout d'une instruction dans le texte, nous sommes bien conscients que, comme l'ordre des instructions a de l'importance, il serait nécessaire de pouvoir en insérer n'importe où dans le texte.
  Cependant, nous considérerons l'hypothèse simplificative suivante : l'ajout d'une instruction dans le texte équivaut à l'ajout d'une ou plusieurs lignes en fin de texte. Une telle organisation est donc également avantageuse pour le sous-mode graphique.
  Remarquons, que tout ajout dans le texte peut provoquer des dépassements de capacité au niveau des lignes (une ligne = 256 caractères). On résoudra ce problème, en créant une nouvelle ligne afin de permettre d'effectuer un ajout correct et complet lié à la modification.

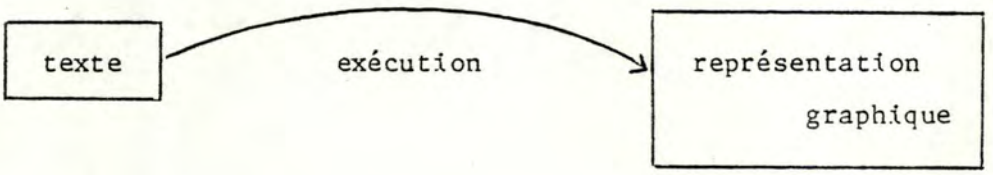
Nous proposerons donc une organisation basée sur la notion de ligne, la structure en instructions n'étant créée lors de l'exécution du texte réalisée lors de l'entrée en sous-mode graphique

4.3. REPRESENTATION GRAPHIQUE D'UN DESSIN

La représentation graphique d'un dessin est le résultat de l'exécution d'un opérateur. Une telle représentation peut être mémorisée de différentes manières. Cependant à ce niveau, nous nous contenterons de parler de la représentation à l'écran et pas de son stockage. Le stockage est un problème inexistant en mode édition-opérateur (on ne garde aucun dessin). Ce problème sera cependant présent dans le mode fichier-dessin. Nous en ferons une étude générale au point 4.6. A l'écran, le dessin sera vu comme un ensemble de points.

4.4. EXECUTION D'UN OPERATEUR

Après avoir parlé du texte d'un opérateur, de sa représentation en mémoire et de la représentation graphique d'un dessin, nous allons parler de l'exécution d'un opérateur qui nous permet de passer d'une représentation à l'autre.



famille

un dessin de cette famille

figure 2

4.4.1. DIFFERENCE ENTRE UN EDITEUR DE TEXTE DE NOTRE MODE EDITION-OPERATEUR.

Lorsque nous réalisons l'édition d'un opérateur, nous réalisons l'édition de son texte. Nous allons voir, qu'il existe une différence entre un éditeur de texte, qui de manière classique réalise l'édition d'un texte et notre mode édition-opérateur.

Un éditeur de texte manipule un texte. Ce texte constitue la donnée. En dehors du texte en cours de traitement, aucun autre texte n'est nécessaire. Dans notre cas, le principe est différent. Bien sûr, dans le mode édition opérateur, il existe un sous-mode texte dont le fonctionnement peut être assimilé à celui d'un éditeur de texte. Mais l'autre sous-mode, le sous-mode graphique nécessite un support graphique. Et qui dit support graphique dit exécution d'un opérateur. D'après la définition que l'on a donnée du texte d'un opérateur, l'exécution de celui-ci nécessite l'appel aux textes des opérateurs utilisés dans sa

définition, et ainsi de suite.

Donc lors d'une édition dans ce sous-mode, en plus du texte de l'opérateur en cours d'édition, on aura besoin:

- du texte des opérateurs nécessaires lors de l'exécution de l'opérateur en cours d'édition, afin d'obtenir le support graphique, lors de l'entrée dans ce mode

- du texte des opérateurs que l'utilisateur désirera utiliser, lors de l'édition dans ce sous-mode

Nous voyons donc où se situe la différence.

Nous poserons comme hypothèse que, lors de l'édition d'un opérateur, tous les textes des opérateurs sont disponibles en mémoire centrale.

4.4.2. PROBLEME : LA SYNTAXE

Un problème important lors de l'exécution d'un opérateur est la syntaxe.

Nous n'avons voulu, lors de la conception de notre éditeur graphique, à aucun moment, contraindre l'utilisateur à vérifier la syntaxe des opérateurs (textes) qu'il édite.

Il pourra donc sortir d'une édition en mode texte, avec un texte dont la syntaxe est incorrecte. Cependant, nous lui fournissons la commande "VS" permettant de déterminer si la syntaxte du texte en cours d'édition est correcte.

Vu qu'aucun contrôle n'est obligatoire, même si le texte de l'opérateur édité est syntaxiquement correct, il se peut que le texte d'un opérateur utlisé dans sa définition, ne le soit pas et ainsi de suite.
Le problème n'est donc pas simple.

Ce problème ne se pose cependant pas à la sortie d'une édition en sous-mode graphique, car pour obtenir le support graphique, il fallait que la syntaxe des opérateurs utlisés (et de l'opérateur en cours d'édition si modification) soit correcte. De plus, c'est le système qui automatiquement fait les modifications correspondant à certaines commandes, dans le texte en édition.
Dans ce cas, à la sortie, le texte sera alors toujours syntaxiquement correct.

En conclusion, on peut donc dire que ce problème va conditionner l'achèvement d'une exécution.

4.4.3. COMMENT REALISER UNE EXECUTION ?

L'exécution d'un opérateur sera constituée de différentes étapes.
Plusieurs éléments sont à prendre en compte:

1. Il faut disposer

   - du texte de l'opérateur que l'on veut exécuter

   - des textes des opérateurs nécessaires à cette exécution (cfr. notion de combinaison)

2. Il faut traiter le problème de la syntaxe

3. Il faut effectuer le passage des paramètres

4. Il faut effectuer le ruissellement des modificateurs afin de déterminer, pour chaque opérateur de base:

   - quelles sont les bonnes valeurs des opérations non transférables à lui associer.

- quel produit de composition des opérations transférables lui associer.

De plus, il faut déterminer quelle est la bonne valeur de l'état ECRAN, valeur définie pour tous les opérateurs de base.

5. Il faut effectuer l'affichage à l'écran de la représentation graphique.

Bien sûr, le fait de rencontrer des problèmes dans le traitement de l'un ou l'autre élément, aura des répercussions sur le traitement des autres.

Nous allons maintenant développer ces différents points.

4.4.3.1. DISPOSER DES TEXTES

Ce problème a été résolu lorsque nous avons posé comme hypothèse, que l'on disposait de tous les textes nécessaires en mémoire centrale (cfr 4.2.2.). Cette hypothèse peut être vue comme fort simplificatrice.

Nous sommes cependant bien conscients que l'on aura besoin de différents supports afin de stocker les textes des opérateurs:

- Certains seront stockés sur bande (= accès séquentiel). On pourrait stocker sur de tels supports, les opérateurs que l'utilisateur ne désire plus utiliser régulièrement (momentanément), et à sa convenance.
  De plus, on pourrait envisager de stocker sur de tels supports, les dessins créés par l'utilisateur.

- Certains seront disponibles en mémoire centrale (par exemple, les opérateurs de base).

- Certains seront stockés sur disquette (= accès direct).

4.4.3.2. PROBLEME DE LA SYNTAXE

Lors de l'exécution, il va falloir repérer les différentes instructions contenues dans le texte de l'opérateur. Vu le fait qu'un opérateur fait appel à d'autres opérateurs dans sa définition, il va donc falloir repérer les instructions de tous les textes nécessaires à l'exécution.

L'exécution va donc nous faire voyager verticalement dans l'arbre de structure.

Afin de déterminer les instructions, le système va scanner le texte de l'opérateur.
Lorsqu'une instruction est détectée et que sa syntaxe est correcte par rapport au langage LG, elle peut être exécutée et ainsi de suite.
Si on détecte une erreur de syntaxe, l'exécution est stoppée définitivement. Cependant, on peut définir des règles permettant de determiner si cette erreur empêche toute poursuite du travail ou non.
Un "warning" pourrait être simplement affiché et le système continue son travail de scannage dans le cas où l'erreur ne serait pas définie comme fatale.

4.4.3.3. PASSAGE DES PARAMETRES

Lorsqu'on a déterminé une instruction, qui de plus est syntaxiquement correcte, nous avons mentionné qu'on l'exécutait. Ceci sous-entend, bien entendu, que l'on effectue le passage des paramètres (conformément aux règles définies dans le langage LG (cfr.3.3.2.)).

Cependant, ce passage n'est pas le seul travail effectué lors de l'exécution; il faut effectuer le ruissellement des modificateurs.

4.4.3.4. RUISSELLEMENT DES MODIFICATEURS

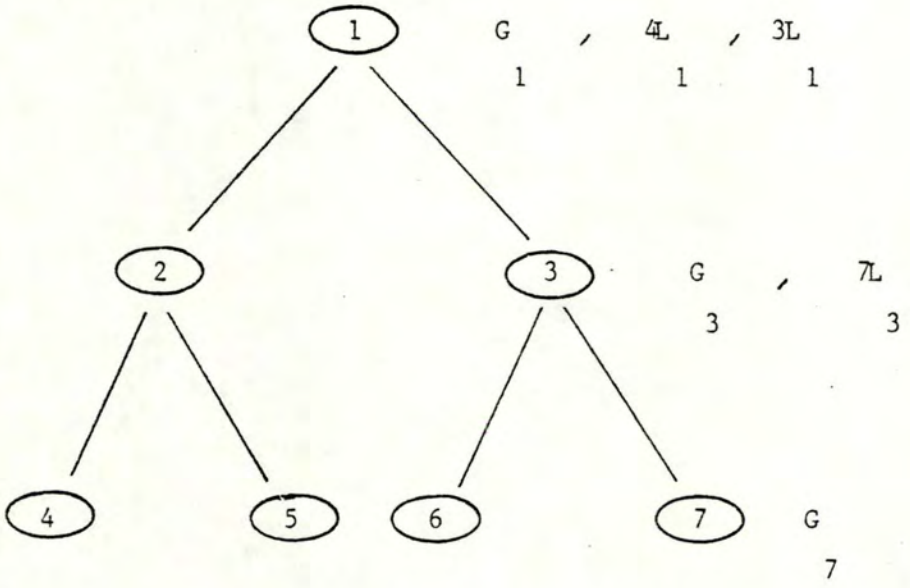
Comme nous l'avons déjà cité au point 3.3.1.2.1., le modificateur est un moyen aisé, fourni à l'utilisateur, pour définir une ou plusieurs opérations particulières agissant à un endroit précis de la structure d'un opérateur, c'est-à-dire sur un constituant déterminé (cfr 3.3.1.2.4.E). De plus, ces opérations influencent le sous-arbre défini à partir de ce constituant dans la structure de l'opérateur, sauf si celui-ci est une feuille.

De ce fait, pour un constituant d'une structure donnée, celui-ci peut être influencé, par des opérations définies à son niveau (les globales), par des opérations définies à un ou plusieurs niveaux supérieurs et le concernant directement (les locales), par des opérations définies également à un ou plusieurs niveaux supérieurs mais, portant sur un constituant racine d'un sous-arbre, auquel notre premier constituant

appartient (influence indirecte).

Par conséquent, on constate qu'à chaque constituant de la structure peut être lié un ensemble d'opérations. Ceci est illustré dans la figure suivante:

Soit la structure d'un opérateur:



où

-Gi est l'ensemble des globaux liés au constituant i.

-AL est le modificateur local défini au niveau
k du constituant k, portant sur le constituant A.

De cette figure, nous pouvons dire que le constituant no7 est influencé par :

- ses globaux(G7)

- les locaux 7L3 de façon directe

- les globaux G1, G3 et les locaux 3L1 de façon indirecte.

Une question se pose : comment toutes ces opérations vont-elles influencer ce constituant ?

Pour y répondre, il suffit de consulter le point 3.3.1.2.4. qui explique les priorités entre opérations.
Rappelons qu'il y a deux types d'opérations :

- les transférables

- les non-transférables

et que pour chaque type, des priorités particulières sont définies, les règles de priorité étant définies plus précisément, en F du point précité.

L'analyse faite précédemment est générale, elle concerne tous les constituants d'une structure. En fait, dans le cadre d'une exécution, l'objectif est de déterminer les opérations associées à chaque opérateur de base, lié aux feuilles de la structure.

Le ruissellement des modificateurs en respectant ces règles permettra de déterminer pour chaque feuille comme nous l'avons déjà cité en début du point 4.4.3., la valeur de chaque opération non-transférable lui étant associée ainsi que le produit de composition des opérations transférables associées.

De plus, il permettra de définir la bonne valeur du fond de l'écran. Cette dernière sera déterminée en fonction des règles de priorité définies au point 3.3.1.2.4.G^3.

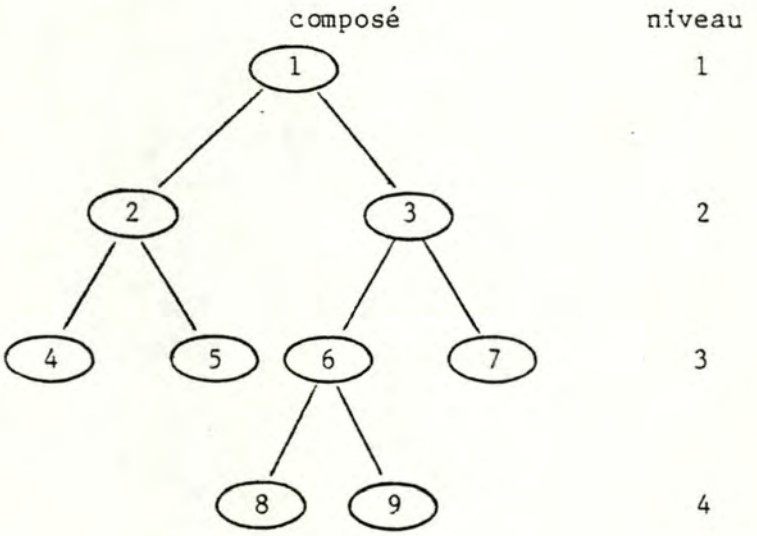
Il est important de signaler que le ruissellement des modificateurs s'effectue au fur et à mesure, après la lecture de chaque instruction, principalement les instructions état et opérateur. De plus, il faut que la syntaxe de chaque instruction soit correcte, sinon le ruissellement sera interrompu et les informations jusqu'alors retenues ne serviront plus à rien.

Nous avons réalisé un algorithme en pseudo-langage, permettant de déterminer pour chaque feuille d'une structure, quelles opérations

transférables et non transférables lui sont associées, ainsi que la valeur du fond de l'écran.
Une description, ainsi qu'une explication de cet algorithme et de son fonctionnement, sont fournies en Annexe H.

4.4.3.5. REPRESENTATION GRAPHIQUE

Considérons la structure d'un opérateur quelconque :



Cet opérateur a dans sa structure 5 appels à des opérateurs de base.

Nous avons donc 5 chemins :

1 2 4
1 2 5
1 3 6 8
1 3 6 9
1 3 7

Lorsque, lors de l'exécution, on rencontre une instruction opérateur, contenant l'appel à un opérateur de base, on sait, d'après la définition d'un opérateur de base, que la cascade d'appels s'arrête, pour ce chemin dans la structure. Une fois l'exécution de l'opérateur de base terminée, il faudra traiter d'autres chemins de l'arbre de structure (s'il en existe).

L'exécution de l'opérateur de base est ce qui va nous permettre d'obtenir la représentation graphique du dessin à l'écran.
Nous n'avons pas encore parlé, jusqu'à présent, de la manière dont est écrit un opérateur de base.
Par manque de temps, et vu le fait que nous avons considéré que ce problème était un problème classique de la littérature graphique, nous n'avons pas approfondi ce problème.
On peut trouver dans la littérature, de multiples algorithmes performants, permettant de tracer une droite ou un cercle par exemple.

Revenons maintenant à notre schéma.
Nous avons déterminé pour cet opérateur, 5 chemins; donc 5 opérateurs de base.
Pour chaque opérateur de base, il faudra appliquer toutes les opérations retenues pour celui-ci lors du ruissellement des modificateurs.

Afin de ne pas faire patienter trop longtemps l'utilisateur à l'écran, nous choisissons de traiter un opérateur de base, et puis directement après faire apparaître à l'écran la représentation graphique correspondante, avant de passer au suivant. La representation graphique va donc se construire au fur et à mesure à l'écran.
Une chose est à effectuer avant de traiter le premier opérateur de base : mettre l'écran dans la couleur définie par le ruissellement des modificateurs.

4.4.3.6. PROPOSITION D'UNE DEMARCHE POUR LA REALISATION DE L'EXECUTION

Nous avons proposé au point 4.4.3.4. et en annexe H, un algorithme permettant la réalisation du ruissellement des modificateurs.
Comme cet algorithme est basé sur la notion d'instruction et que l'exécution d'un opérateur l'est également (repérer une instruction et l'exécuter), nous pourrions prendre comme base pour réaliser celle-ci, cet algorithme.
Il nous restera donc à tenir compte, dans l'algorithme, du problème de la syntaxe (4.4.3.2), du passage des paramètres (4.4.3.3) et de l'affichage (4.4.3.5), le point 4.4.3.1. étant résolu par les hypothèses que nous avons posées.
Le problème de la syntaxe sera abordé lors de la lecture de l'instruction dans l'algorithme.
Lors de l'analyse de ce problème, le système scannera le texte afin de

vérifier la syntaxe de celui-ci et donc par la même occasion, de déterminer les instructions qui pourront alors être exécutées au fur et à mesure.

En ce qui concerne le passage des paramètres, il pourra également être inséré lors de la lecture d'une instruction, et ce, conformément aux règles définies dans le langage LG (cfr3.3.2).

A la fin de la phase de lecture d'une instruction, celle-ci sera syntaxiquement correcte (sinon on aurait stoppé en cours de traitement) et chaque paramètre aura été traité.

Il restera à effectuer l'affichage.
Celui-ci se fera uniquement après la lecture d'une instruction opérateur correspondant à un opérateur de base.
Ayant toutes les informations nécessaires concernant les paramètres, la valeur des opérations non-transférables et le produit de composition des opérations transférables (obtenus par le ruissellement des modificateurs), nous pourrons alors exécuter l'opérateur de base.

4.5. EDITION D'UN OPERATEUR

Comme nous l'avons expliqué précédemment, le mode édition-opérateur est composé de deux sous-modes différents

- le sous-mode texte

- le sous-mode graphique

De plus, l'utilisateur peut passer d'un sous-mode à l'autre. Ce passage nécessite un "changement d'appui", c'est-à-dire que l'utilisateur dispose en sous-mode texte, d'un appui texte et en sous-mode graphique, d'un appui graphique.

Schématiquement, on a:

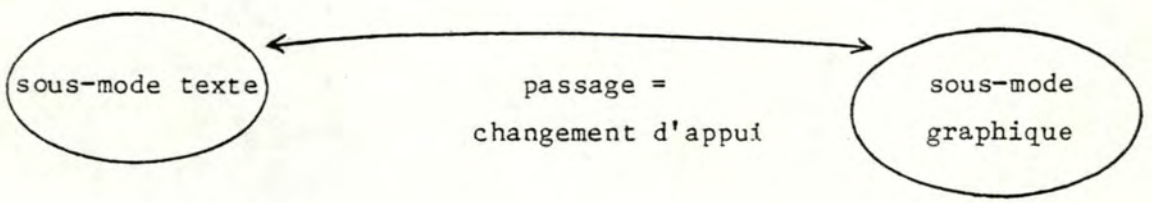


figure 3

Nous allons donc dans ce point, développer, dans un premier temps, chaque sous-mode séparément et puis, traiter le passage de l'un à l'autre.

4.5.1. LE SOUS-MODE TEXTE

Rappel : grâce à ce mode, l'utilisateur va pouvoir éditer le texte d'un opérateur via un mini-éditeur de texte.

Schématiquement, on obtient :

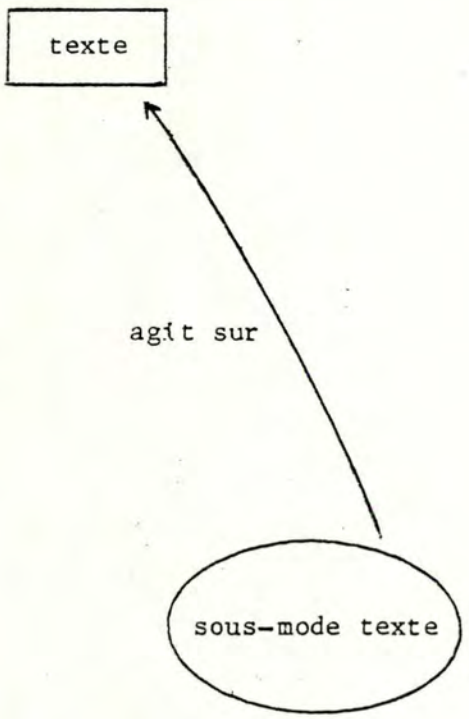


figure 4

Lors de l'entrée dans ce sous-mode, l'utilisateur doit disposer du texte de l'opérateur qu'il désire éditer.

Comme indiqué au point 3.4.6.1., nous n'avons pas distingué explicitement le cas de la création d'un opérateur, du cas de la modification d'un opérateur.

Bien évidemment, dans le cas d'une création au niveau du schéma, le texte n'existe pas. Il se crée au fur et à mesure .

L'entrée dans ce sous-mode peut être réalisée de deux manières:

- via la commande EDITEXT
- via le passage entre le sous-mode graphique et le sous-mode texte.

Le passage entre les deux sous-modes fera l'objet d'une analyse particulière (cfr. 4.5.3.).
Vu que l'on peut trouver une analyse complète de la commande EDITEXT en annexe G au point I.1.1., il suffit de se référer à cette annexe, afin de trouver une explication de l'entrée dans ce sous-mode via cette commande.

Le principe de fonctionnement du sous-mode texte est un principe classique. On peut dire qu'il se résume au fonctionnement d'un éditeur de texte. De plus, il est nécessaire de disposer pour le texte, d'une structure basée sur la notion de ligne. Nous en avons parlé au point 4.2.1.
En considérant que le noeud central de ce sous-mode peut être assimilé à un éditeur de texte et, en fonction du fait qu'un éditeur de texte a déjà fait l'objet de plusieurs mémoires dans cet institut, nous n'avons pas approfondi sa réalisation. Ce sous-mode ne constitue pas la particularité de notre système, le sous-mode graphique et le passage étant plus, des éléments propres à notre approche. Cependant nous pouvons parler de deux choses extérieures à cet éditeur : le problème des arbres de structure et de la vérification syntaxique.

Les arbres de structure des opérateurs

Lors de l'édition du texte d'un opérateur dans ce sous-mode, l'utilisateur va, par définition d'un opérateur, se servir d'autres opérateurs. De plus, il pourra utiliser ce que l'on a appelé : les modificateurs.
A moins que de connaître de mémoire, la structure des opérateurs dont il se sert (ce qui n'est possible que pour un nombre très limité d'entre-eux), il a besoin de pouvoir disposer, à tout moment dans son travail, de structures afin de pouvoir utiliser les modificateurs.
Il faudrait donc pour les opérateurs, disposer de leur arbre de structure en accès rapide.
Cependant, un problème se pose. Disposer de l'arbre de structure, implique que la syntaxe de l'opérateur soit correcte.
Comme une édition en sous-mode texte est réalisée sans condition de vérification de syntaxe, cette structure ne pourra être créée que lors de l'exécution de l'opérateur, et si cette exécution se déroule sans erreur (on ne disposera donc pas de la structure de l'opérateur en cours d'édition en mode texte), on ne pourra donc fournir la structure d'un opérateur que s'il a déjà été exécuté, ce qui implique aussi, qu'elle

sera disponible si la dernière édition de cet opérateur a été réalisée en sous-mode graphique (car la condition d'entrée dans ce sous-mode est, que la syntaxe de l'opérateur qu'on veut modifier soit correcte, afin d'obtenir le support graphique).
De plus, si on dispose de la structure d'un opérateur, toute nouvelle édition en sous-mode texte va rendre invalide cette structure. Il faudra donc rendre cette information non disponible.
Cependant si la sortie de l'édition est une sortie du sous-graphique, on pourra disposer de la nouvelle structure.

Donc en conclusion on peut dire que, comme on n'impose pas qu'à la sortie du sous-mode texte la syntaxe du texte édité soit correcte, on peut lors d'une édition en sous-mode texte

- utiliser un opérateur dont la syntaxe est incorrecte

- ne pas disposer des arbres de structure de tous les opérateurs pour lesquels on désire ce renseignement.

La vérification syntaxique (VS)

La vérification syntaxique est un outil que nous fournissons à l'utilisateur, afin de pouvoir vérifier la syntaxe de l'opérateur en cours d'édition.
Cette vérification se limitera au texte de l'opérateur en cours d'édition, sans vérifier le texte des opérateurs appelés dans le texte de celui-ci.

Comme nous l'avons vu précédemment au point 4.4. (exécution d'un opérateur) et au niveau des arbres de structure, la syntaxe est un problème important. Afin de réduire la probabilité de traiter un opérateur dont la syntaxe est erronnée, on aurait pu envisager de fournir à l'utilisateur une commande plus complète que VS, en ne la limitant pas au seul texte de l'opérateur en cours d'édition.

Elle aurait permis de vérifier également les textes de la cascade d'opérateurs utilisés dans les définitions d'opérateurs. Une autre idée, allant dans le même sens, mais qui cependant pourrait être nettement plus efficace, serait de profiter des temps morts entre les caractères tapés par l'utilisateur afin de vérifier directement et automatiquement la

syntaxe, et de fournir des messages d'erreurs au fur et à mesure du travail.

Une telle méthode permettrait de réduire la quantité de travail inutile fournie (affichages), si l'on rencontre ultérieurement une erreur de syntaxe, chose importante vu que nous sommes sensés travailler sur microordinateur.

4.5.2 LE SOUS-MODE GRAPHIQUE

Rappel : le sous-mode graphique permet à l'utilisateur d'éditer le texte d'un opérateur à l'aide d'un support graphique. Via des commandes, l'utilisateur va agir sur la représentation graphique d'un dessin issu de la famille décrite par l'opérateur en cours d'édition. De plus, certaines commandes vont avoir des répercussions sur le texte de l'opérateur en cours d'édition.

Schématiquement, on obtient :

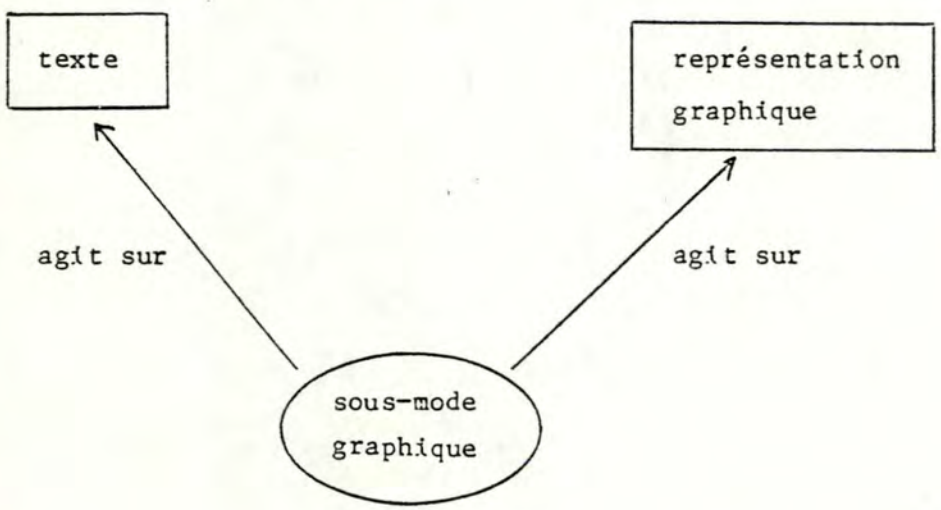


figure 5

Bien qu'au niveau conceptuel, ce sous-mode est caractérisé par un appui graphique, au niveau des principes de réalisation, la présence simultanée du texte de l'opérateur en cours d'édition et de la représentation graphique du dessin support de l'édition est néecessaire.

Nous développons ici les points suivants :

- l'entrée dans le sous-mode graphique
- la manière dont le sous-mode graphique agit
    * sur le texte de l'opérateur en cours d'édition
    * sur la représentation graphique du dessin support de l'édition

4.5.2.1. L'ENTREE DANS CE SOUS-MODE

L'entrée dans ce sous-mode peut être réalisée de deux manières :

- via la commande EDITGRAPH
- via le passage entre le sous-mode texte et le sous-mode graphique

Nous ne parlerons ici que de l'entrée via la commande "EDITGRAPH", car l'autre entrée fera l'objet d'une analyse particulière (cfr 4.5.3).

Lors d'une entrée dans ce sous-mode via la commande "EDITGRAPH", on peut se trouver dans deux situations différentes :

- soit on édite un opérateur non existant, ce qui équivaut à une création
- soit on édite un opérateur existant, ce qui équivaut à une modification

Dans la première situation , l'édition démarrera avec un écran vierge et le texte de l'opérateur contiendra uniquement la ligne suivante :

OPERATEUR (nom opérateur), avec (nom opérateur) étant
le nom de l'opérateur que l'utilisateur désire créer.

Donc dans cette situation, la structure en instruction nécessaire dans le sous-mode graphique, sera limitée à l'en-tête de l'opérateur. De plus, l'arbre de structure de l'opérateur en cours d'édition sera inexistant, tout comme la liste des variables de cet opérateur.

Dans la deuxième situation, différentes choses sont à effectuer ;

a) exécuter l'opérateur que l'on veut éditer afin d'obtenir le dessin support graphique de l'édition

b) construire la structure en instructions liée au texte et nécessaire dans ce sous-mode

c) construire l'arbre de structure de l'opérateur en cours d'édition et du dessin support de l'édition

d) fournir la liste des variables de l'opérateur en cours d'édition

Nous allons analyser maintenant, les 4 points de cette deuxième situation

a) exécution

Nous avons développé ce problème au point 4.4.3. Cependant, il faut être conscient que si un problème apparaît lors de cette exécution (syntaxe), il n'est donc pas possible de rester dans ce sous-mode. Par ce fait, il est impossible de satisfaire les points b), c) et d). Cependant, lors de l'exécution , on peut être amené à effectuer des affichages inutiles à l'écran, si l'exécution est stoppée à cause d'un problème.

On pourrait envisager, qu'afin de ne pas effectuer un travail inutile en cas d'erreur de syntaxe, de ne pas afficher à l'écran la représentation graphique issue de chaque opérateur de base au fur et à mesure. Si on se contentait de mémoriser tous les renseignements associés à chaque opérateur de base, et de n'afficher les représentations graphiques qu'à la fin, uniquement si aucune erreur n'a été rencontrée, on pourrait ne pas perdre inutilement du temps en affichage qui ne servirait à rien. Cependant, une telle solution pourrait procurer à l'utilisateur, un temps d'attente indésirable, car il ne verrait rien se dessiner à l'écran, l'exécution des opérateurs de base (affichage) ne s'effectuant qu'à la fin, si le travail de préparation n'a rencontré aucun problème.

b) structure en instruction

Au fur et à mesure de l'exécution, on va construire la structure en

instruction. Le mécanisme serait le suivant. Nous avons expliqué que lors de l'exécution, il fallait repérer les différentes instructions afin de pouvoir les exécuter. Pour cela, le système scanne le texte, détermine les instructions et les exécute si leur syntaxe est correcte. Il est donc très simple à ce moment de retenir la localisation des instructions dans le texte (rappelons que nous avons proposé au point 4.2.2. de baser l'organisation du texte en mémoire centrale sur la structure en ligne). Cette structure ne sera donc disponible que si l'exécution est réalisée jusqu'au bout, sans problème de syntaxe.

c) arbre de structure de l'opérateur en cours d'édition et du dessin support de l'édition

En considérant le même principe que pour la structure en instructions, il est aisé lors de l'exécution, chaque fois que l'on a repéré une instruction opérateur, de retenir son qualificateur afin de construire l'arbre. Un problème dans l'exécution (syntaxe) aura le même effet que dans le cas précédent. Comme nous l'avons expliqué au niveau de l'analyse fonctionnelle, il y aura correspondance entre les deux arbres, les noms des noeuds étant identiques, excepté pour la racine. Donc, en construisant une des deux structures, on obtient automatiquement l'autre.

d) fournir la liste des variables de l'opérateur en cours d'édition

Lors de la détermination des instructions, c'est-à-dire lors du scanning du texte, on peut chaque fois que l'on rencontre un paramètre, une variable d'une instruction d'affectation, qui ne correspond pas à un paramètre, construire une liste avec pour chacune d'elle, son nom et sa valeur. Cette liste sera mise à jour au fur et à mesure du scanning du texte afin de délivrer à la fin de l'exécution, une liste contenant la valeur finale de chacun de ces éléments. De plus, cette liste contiendra pour chaque variable un renseignement permettant de savoir si la variable dont l'utilisateur désire se servir, apparaît ou non dans les paramètres formels de l'opérateur en cours d'édition. Cette liste ne contiendra que les variables du texte de l'opérateur en cours d'édition car, ce sont ces seules variables qui intéressent l'utilisateur lors d'une édition en sous-mode graphique.

4.5.2.2. COMMENT LE SOUS-MODE GRAPHIQUE AGIT-IL SUR LE TEXTE DE L'OPERATEUR EN COURS D'EDITION ?

Toutes les commandes de l'utilisateur définies dans ce sous-mode ne vont pas avoir des répercussions sur le texte de l'opérateur en cours d'édition. Seules, les commandes des classes "dessin" et "définition de paramètres" vont avoir des répercussions, qui ont été définies en annexe G. Le fait de disposer de la structure en instruction va avoir ici toute son importance. Si on assimile l'entête de l'opérateur à une instruction, les changements à apporter dans le texte se résumeront à :

- dans le cas d'une commande de la classe "définition de paramètres", on va, en respectant les règles de répercussions, changer l'instruction en "en-tête". De plus, il faudra modifier la liste des variables.

- dans le cas d'une commande de la classe "dessin" :

  * si on a affaire à une commande de traçage :

    Si on considère que l'ajout d'une instruction opérateur correspond à l'ajout d'une ligne en fin du texte, alors ce changement sera aisé (cfr point 4.2.2).
    il faudra mettre à jour la structure en ligne et en instruction. Donc de plus, il faudra ajouter des noeuds à l'arbre de structure de l'opérateur en cours d'édition ainsi que dans celui du dessin support de l'édition.

  * si on a affaire à une commande état :

    Il faudra connaître l'emplacement de la première instruction opérateur dans le texte ainsi que l'absence ou non d'instructions état avant cette première instruction opérateur. Ceci pourra être fait via la structure en instruction. Donc, grâce à cette structure en instruction, il ne faudra pas scanner tout le texte, afin de déterminer l'emplacement exact de la modification à apporter. Il faudra également mettre àjour la structure en ligne et celle en instruction.

  * si on a affaire à une commande opération :

    Grâce aux listes de noms de dessins, on va pouvoir

déterminer quelle instruction opérateur doit être modifiée ou supprimée. Il faudra aussi mettre à jour la structure en lignes et celle en instructions et, si nécessaire (cas supprimer), changer l'arbre de structure de l'opérateur en cours d'édition ainsi que celui du dessin support de l'édition.

En conclusion, on peut dire que cet aspect de la figure 5, ne posera pas de gros problèmes au niveau réalisation.

4.5.2.3. COMMENT LE SOUS-MODE GRAPHIQUE AGIT-IL SUR LA REPRESENTATION GRAPHIQUE DU DESSIN SUPPORT DE L'EDITION ?

On dispose donc dans ce sous-mode d'un appui graphique support de l'édition.
En plus d'agir sur le texte de l'opérateur en cours d'édition, ce sous-mode agit sur la représentation graphique du dessin support de l'édition et ce, via certaines commandes (cfr annexe G, I.2).
Cet aspect de la figure 5 est plus délicat que l'autre.

Deux attitudes sont possibles vis-à-vis de ce problème :

- effacement complet de l'écran et réexécution complète de l'opérateur en cours d'édition après modification du texte.

- effacement localisé et réexécution partielle après modification du texte.

4.5.2.3.1. EFFACEMENT ET REEXECUTION COMPLETE

Cette solution consisterait à, après avoir apporté les modifications nécessaires dans le texte, réexécuter celui-ci afin d'obtenir à l'écran, le support graphique avec la modification apportée. Cela nécessite bien évidemment que ce qu'il y avait à l'écran, ait auparavant été effacé. Cette solution est une solution simple, qui permet d'éviter les problèmes de la deuxième solution (effacement local). Cependant, il faut être conscient du fait que, cette solution peut être coûteuse en temps. Il faut effectuer à nouveau tout le ruissellement des modificateurs : chose parfois inutile si on s'est contenté, par exemple, de simplement ajouter une instruction opérateur. Il aurait été plus simple, de seulement exécuter l'opérateur utlilisé auquel correspond cette instruction.

4.5.2.3.2 EFFACEMENT ET REEXECUTION LOCALE.

Cette solution est basée sur la remarque exposée dans le point précédent. L'idée est, de ne pas réexécuter tout l'opérateur, mais de simplement apporter à la représentation graphique de l'écran, un changement localisé. Le cas de la commande de traçage est particulier car, il suffit d'exécuter l'opérateur et d'ajouter, à ce qu'il y avait déjà à l'écran, la représentation graphique y correspondant. Les commandes opérations nécessitent des modificateurs de ce qui existait à l'écran. Trois étapes sont alors nécessaires :

- un effacement localisé
- une modification dans le texte
- une exécution de l'opérateur concerné par les modificateurs.

a) effacement localisé:

Ce problème n'est pas simple. Toute modification à apporter est en fait une modification effectuée sur un constituant de la structure de l'opérateur en cours d'édition. (On peut bien sûr, apporter un changement à la racine (composé) de cette structure). On peut donc déterminer quel noeud de la structure est touché par la modification, afficher le constituant tel qu'il était avant modification dans la couleur de fond de l'écran (afin de le faire disparaître).

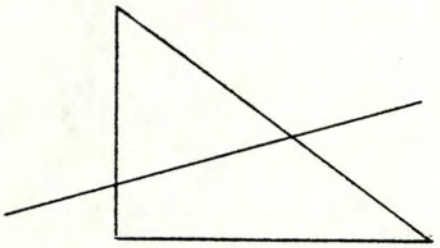
Cependant une telle solution va engendrer les problèmes suivants :

- la méthode utilisée afin de réaliser l'effacement va poser un problème d'exactitude. Si on considère les points de l'écran allumés ou éteints, on ne saura pas nécessairement éteindre chaque point (pixel) allumé, ce qui va rendre l'effacement imprécis et laisser des points non désirés à l'écran; d'où on obtiendra une représentation graphique à l'écran incorrecte. Par une telle méthode, et en multipliant les modificateurs, la représentation graphique à l'écran s'éloignera plus ou moins rapidement de la représentation réelle.
  Cependant, on peut considérer l'écran en sous-mode graphique,

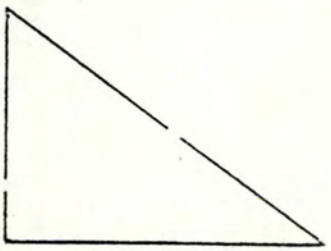
comme étant un brouillon, pour lequel une grande précision n'est pas une condition indispensable. Il serait alors nécessaire de rafraîchir l'écran régulièrement afin d'obtenir une précision moyenne acceptable, et ainsi se rapprocher de la représentation graphique exacte.

- En faisant "disparaître" (rendre de la même couleur que le fond de l'écran) les points appartenant à un constituant, en plus du problème de précision, nous risquons de faire "disparaître" des points qui appartiennent à un autre constituant et qui étaient communs aux deux.

Exemple



Si on désire supprimer la droite, on va obtenir pour le triangle, la situation suivante :



On va donc être amené à gérer des intersections afin de ne pas perdre trop de points.

- Bien que cette solution possède l'avantage de gain de temps par rapport à la précédente, elle peut rendre les représentations graphiques à l'écran incorrectes (ce qui n'est pas toujours trop ennuyeux). Il faut donc faire un compromis entre ces deux aspects.

b) modification du texte

Pour cela, voir en annexe G au point I.2, où on peut trouver les répercussions de chaque commande sur le texte d'un opérateur.

c) exécution

Afin de ne pas tout réexécuter, on exécute seulement l'appel à l'opérateur contenu dans l'instruction opérateur correspondant au constitutant concerné. Il faut bien évidemment tenir compte de toutes les opérations le concernant (déterminées par le ruissellement des modificateurs).

4.5.2.3.3.APPROFONDISSEMENT DE LA SOLUTION QUI CONSISTE EN UN EFFACEMENT ET UNE REEXECUTION LOCALE.

a)Comment réaliser cette solution ?

Rappelons qu'à ce niveau, nous disposons de deux représentations, l'une représentant la famille (le texte de l'opérateur), l'autre un dessin de cette famille (représentation graphique à l'écran).
Comment pourrions-nous à partir de celles-ci, effectuer les trois étapes nécessaires pour réaliser l'effet attendu des commandes opérations à savoir :

- l'effacement localisé

- la modification dans le texte

- une exécution de l'opérateur concerné par la modification.

Tout d'abord, l'effacement localisé consistera en un parcours du texte de telle manière que l'on puisse effectuer un ruissellement partiel. En fait, comme l'on désire n'effacer qu'une seule partie de la représentation graphique d'un dessin à l'écran, toutes les informations concernant le ruissellement du dessin entier ne nous interessent pas.
Pour ce faire, il faudra effectuer un parcours sélectif dans les textes des opérateurs nécessaires à la réalisation de ce ruissellement partiel.

Ce parcours s'effectuera en deux parties : un parcours jusqu'à l'instruction opérateur correspondant au constituant concerné par la modification, ensuite le parcours du sous-arbre dont la racine est ce constituant c'est-à-dire des textes des opérateurs liés à chaque constituant de ce sous-arbre.

Nous obtenons ainsi, pour chaque feuille du sous-arbre, c'est-à-dire pour chaque opérateur de base, le produit de composition des opérateurs transférables et les opérations non transférables associées à chacun d'entre eux.

Il suffit maintenant de calculer les transformations que le produit de composition effectue sur chaque opérateur de base et modifier la valeur de la couleur de chacun comme étant celle du fond de l'écran. Ensuite effectuer l'affichage; celui-ci aura pour effet d'effacer la partie de

dessin représentant le constituant sur lequel porte la modification (l'opération).

L'étape suivante sera la modification du texte de l'opérateur. Pour cette étape, nous nous référons à ce qui a été décrit en annexe G.

Enfin, on refait la même chose que pour l'effacement avec le texte ainsi modifié, sauf en ce qui concerne la couleur de chaque opérateur de base, auquel il ne faudrait pas appliquer la couleur du fond de l'écran, ce qui représentera la troisième étape.

Cette démarche paraît longue, du fait que trois parcours des textes sont nécessaires pour effacer, modifier le texte, le réexécuter pour afficher le résultat de la modification (de l'opération).

b) Passage par une représentation intermédiaire

Une autre démarche serait, d'utiliser une représentation intermédiaire afin d'éviter trop de parcours du texte, lors de la réalisation du ruissellement des modificateurs pour l'effaçage, la réexécution. Celle-ci se crée lors de l'exécution de l'opérateur, dès que l'on entre dans le sous-mode graphique.

En fait, pour chaque feuille de la structure de l'opérateur que l'on exécute, c'est-à-dire pour chaque opérateur de base, on retiendra son nom, les paramètres actuels, le produit de la composition des opérations transférables et la valeur des opérations non transférables lui correspondant. De plus, à chaque opérateur de base, on fait correspondre une liste de qualificateurs, dont chaque élément de la liste correspondant au nom des noeuds successifs hormis la racine du chemin allant, dans la structure de l'opérateur de la racine à la feuille liée à l'opérateur de base. Ceci constituant la représentation intermédiaire.

Disposant de cette représentation, comment allons-nous réaliser l'effet des commandes opérations ?

Nous disposerons d'une liste de qualificateurs, au départ d'une commande de ce type. Celle-ci nous permettra de trouver les opérateurs de base concernés, c'est-à-dire celui ou ceux dont la liste de qualificateurs commence par la liste de qualificateurs de la commande.

Une telle solution possède les désavantages suivants :

- elle occupe beaucoup de place en mémoire due à la redondance des informations stockées (liste de qualificateurs pour chaque opérateur de base).

- elle demande beaucoup de temps, pour la recherche des opérateurs de base, concernés par la modification, due à un parcours de tous les opérateurs de base, liés aux feuilles de la structure de l'opérateur que l'on désire modifier.

Une autre solution pourrait permettre de réaliser cette recherche de façon plus rapide tout en prenant moins de place. Celle-ci ne consisterait plus à retenir pour chaque opérateur de base, la liste des qualificateurs lui étant associée mais plutôt de se servir de la structure de l'opérateur.
En ajoutant à chaque feuille de cette structure, l'adresse des informations liées à l'opérateur de base associé à chacune d'elles (ces informations étant constituées du nom de l'opérateur de base, des paramètres actuels, du produit de composition et des valeurs couleur, trait), nous ferons une économie de place et de temps de recherche. Afin de retrouver les opérateurs de base concernés par la modification décrite dans la commande opération, il suffit de parcourir l'arbre jusqu'au noeud touché par cette modification et par la suite, de parcourir le sous-arbre dont la racine correspond à ce noeud jusqu'aux feuilles qui fourniront l'adresse des informations liées aux opérateurs de base nous intéressants.

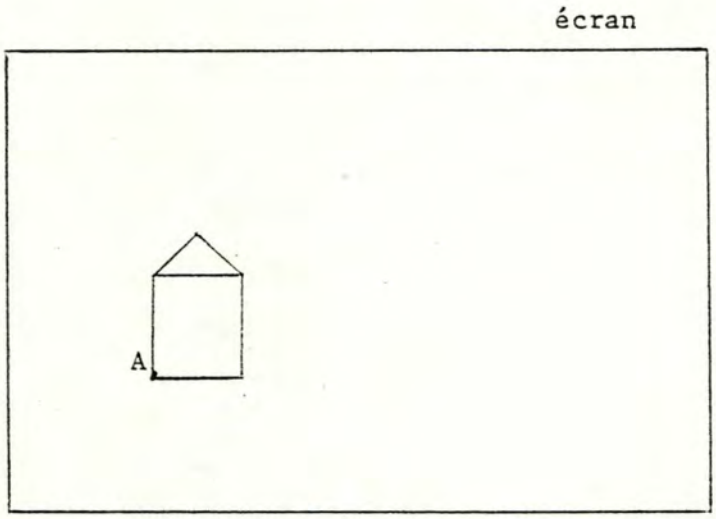
Dès que l'on a trouvé ces opérateurs, deux situations sont possibles :

1) On effectue l'effaçage, la modification du produit de composition des opérations transférables ou de la valeur des opérations non- transférables et la réexécution pour chaque opérateur de base.
C'est-à-dire que l'effacement et l'affichage suite à la réexécution, de la partie du chemin que l'on désire modifier s'effectue au fur et à mesure, morceau par morceau.

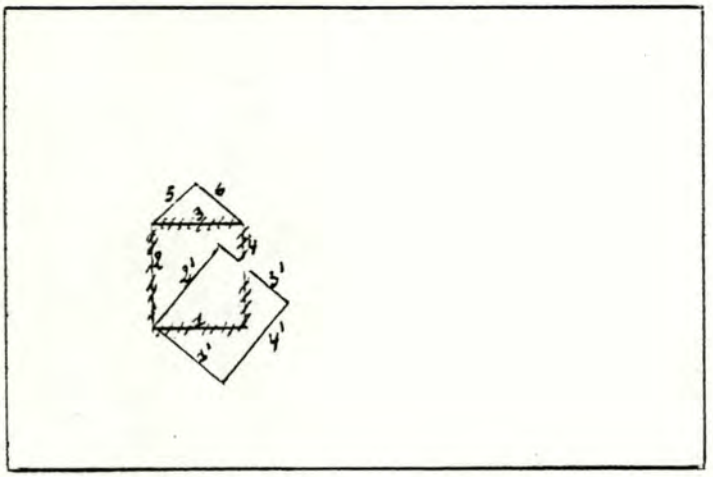
En fait, on se concentre sur un opérateur de base, on l'exécute tel quel pour effacer sa représentation graphique à l'écran (avec couleur = couleur de fond de l'écran), ensuite on modifie le produit de composition

des opérateurs transférables ou la valeur liée aux opérations non-transférables pour directement exécuter l'opérateur de base avec ses nouvelles transformations.

Rappelons que, la modification du produit de composition ou de la valeur, s'effectuera en respectant les règles de priorité qui ont été énoncées dans l'analyse fonctionnelle au point 3.3.1.2.4.
Cependant une telle solution présente un inconvénient.
Pour présenter celui-ci, regardons la figure suivante où l'on retrouve la représentation graphique d'une maison.



Supposons que l'on y applique une transformation du genre rotation à partir du point A, et d'un angle de 45 degrés.
Supposons que cette maison ne soit faite que de segments de droites que nous numérotons de 1 à 6.



Appliquons notre méthode : - la droite 1 descend de 45 degrés, c'est-à-dire qu'elle est d'abord effacée, ensuite réaffichée en respectant la rotation de 45 degrés demandée (1'). Il en est de même pour les droites 2, 3, 4, (2', 3', 4').

Arrêtons-nous quelques instants sur la quatrième droite, on remarquera que lorsque l'on efface celle-ci, on effacera également quelques points de la nouvelle droite 3' et de façon irrémédiable.

Ceci aura pour inconvénient que, la précision du dessin ne sera pas excellente et qu'en fonction du nombre de fois que cette situation se présentera, elle risque encore de diminuer. C'est pour cette raison que la deuxième solution que nous allons exposer nous paraît plus intéressante.

2) D'abord, on effectue l'effacement, c'est-à-dire l'exécution des opérateurs de base avec la couleur équivalente à celle du fond de l'écran pour chaque opérateur de base.
Ensuite, après modification du produit de composition, les opérations non-transférables ou de la valeur des opérations non transférables pour chaque opérateur de base, on réexécute ceux-ci.

C'est-à-dire que, l'effacement de la partie du dessin que l'on désire modifier s'effectue d'abord entièrement, et qu'ensuite on réaffiche la même partie mais ayant subi les modifications.
Cependant, une telle solution risque de provoquer un temps d'attente assez prolongé entre l'effacement et l'affichage de la partie modifiée, ceci dû au fait qu'il faut :

- modifier les produits de composition des opérations transférables et les valeurs, des opérations non-transférables liées aux opérateurs de base concernés par la modification;

- appliquer les produits de composition aux opérateurs de base.

c). Le produit de composition

Nous allons maintenant analyser comment réaliser les deux actions suivantes :

- modifier le produit de composition des opérations transférables lié à un opérateur de base;

- appliquer le produit de composition à un opérateur de base.

c.1. modification du produit de composition

En fait, modifier le produit de composition des opérateurs transférables, s'effectue de la manière suivante :
A partir de la liste des qualificateurs donnée dans la commande opération, on peut déterminer le no de niveau sur lequel porte cette opération ainsi que évidemment le no du niveau de provenance qui est ici en l'occurrence le niveau 2 puisque toute commande opération définit un modificateur de niveau 2.
Comme les informations concernant le produit de composition sont rangées, dans l'algorithme de ruissellement des modificateurs défini en annexe H, en un ensemble d'enregistrements ayant pour chacun d'entre-eux la forme suivante :

NN	Valeur de l'opération	type d'opération	NN provenance

Le no de niveau "NN" du noeud sur lequel porte l'opération, suivi de la valeur de l'opération du type de celle-ci et enfin du no de niveau dont elle provient.
De plus, comme ces enregistrements sont rangés par ordre croissant sur le no de niveau "NN", et pour même NN, par ordre croissant sur NN provenance.
Nous pouvons dès lors procéder à l'insertion de l'opération transférable au bon endroit dans cette suite d'enregistrement, qui sont la base du produit de composition.

L'insertion s'effectue de la manière suivante :

On insérera dans la suite d'enregistrement, l'opération à la fin de ceux ayant comme "NN" le no de niveau sur lequel porte cette opération et comme "NN provenance" le no de niveau 2.
A partir de ces enregistrements, le produit de composition sera construit au fur et à mesure selon l'ordre dans lequel apparaissent ceux-ci.

c.2. application du produit de composition sur un opérateur de base

Expliquons maintenant comment cette application du produit de composition se fera, sur un opérateur de base, en considérant une opération comme une transformation.

c.2.1. Les transformations

Tout d'abord, précisons que nos principes de transformations ne travailleront qu'en deux dimensions et que nous limiterons le développement de ceux-ci aux opérations de base translation, rotation, le changement d'échelle.

Ajoutons, que toute transformation se fera à partir de coordonnées de points, c'est-à-dire que dans l'exemple du segment de droite, nous considérons les deux points extrêmes de celui-ci et non tous les points du segment.

Il suffira alors de retracer un segment de droite joignant ces deux points, afin d'obtenir la représentation graphique divisée à l'écran comme information de base à la transformation que l'on désire appliquer.

Présentons maintenant les différentes transformations et leur forme.

-la translation

La forme de la transformation de type translation est la suivante :

$$x' = x + Tx$$

$$y' = y + Ty$$

où (x,y) représente les coordonnées d'un point

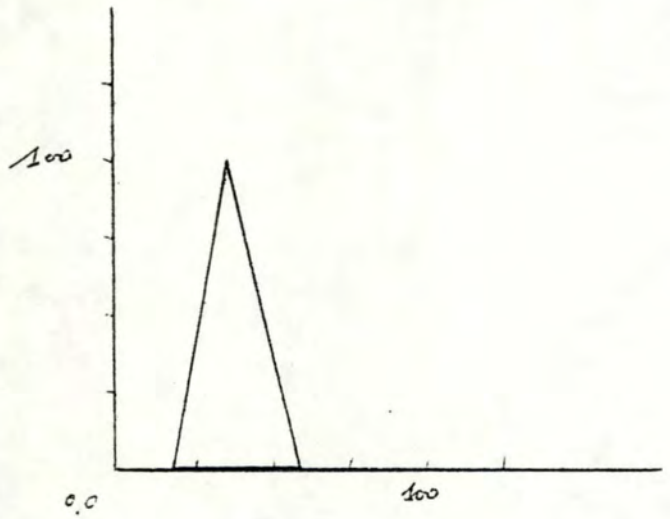
et (x',y') les nouvelles coordonnées.

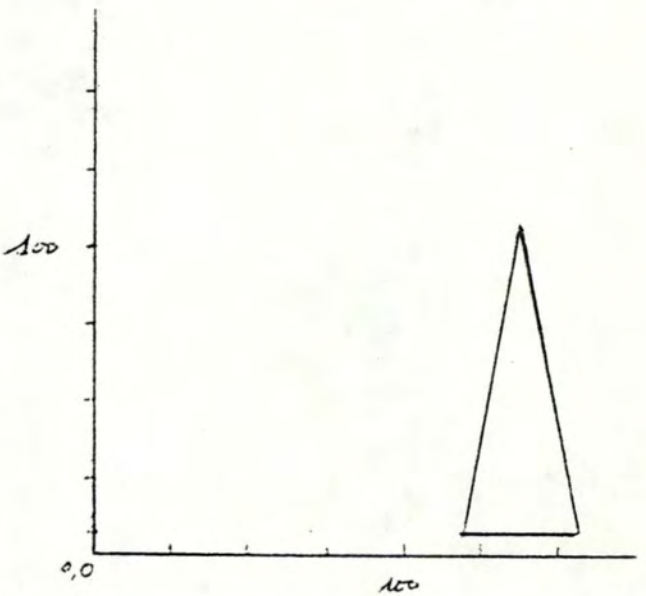
Par exemple, considérons un triangle défini par les trois points suivants : (60, 0), (40,100), (20,0).

Appliquons à celui-ci une translation de 100 unités à droite et 10 unités vers le haut. (Tx = 100, Ty = 10).

Les nouvelles coordonnées sont : (160,10), (140,110), (120,10).

L'effet est montré à la figure suivante :





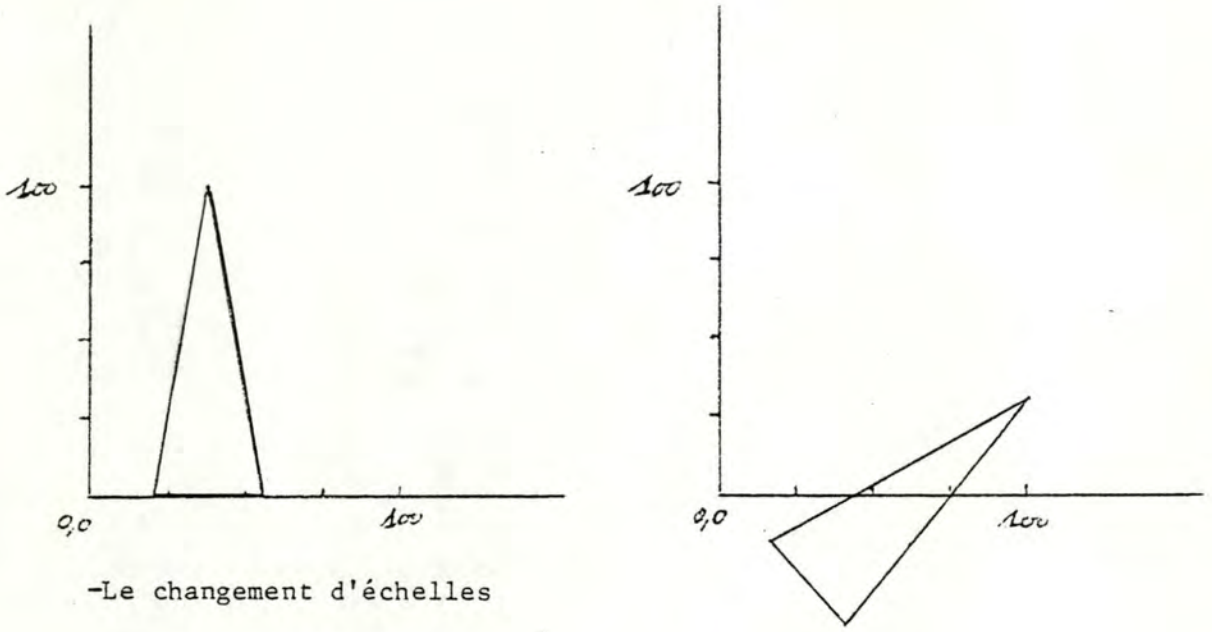
-la rotation

Pour effectuer une rotation du point (x, y) d'un angle de θ degré dans le sens des aiguilles d'une montre par rapport à l'origine du système de coordonnée, nous pouvons écrire :

$$x' = x \cos \theta + y \sin \theta$$
$$y' = -x \sin \theta + y \cos \theta$$

Par exemple, le triangle (20,0), (60,0), (40,100) subissant une rotation de 45 degrés dans le sens des aiguilles d'une montre par rapport à l'origine deviendra :

(14.14, -14.14),(42.43, -42.43), (98.99, 42.43) comme montré à la figure suivante :



-Le changement d'échelles

La forme du changement d'échelle sera la suivante :

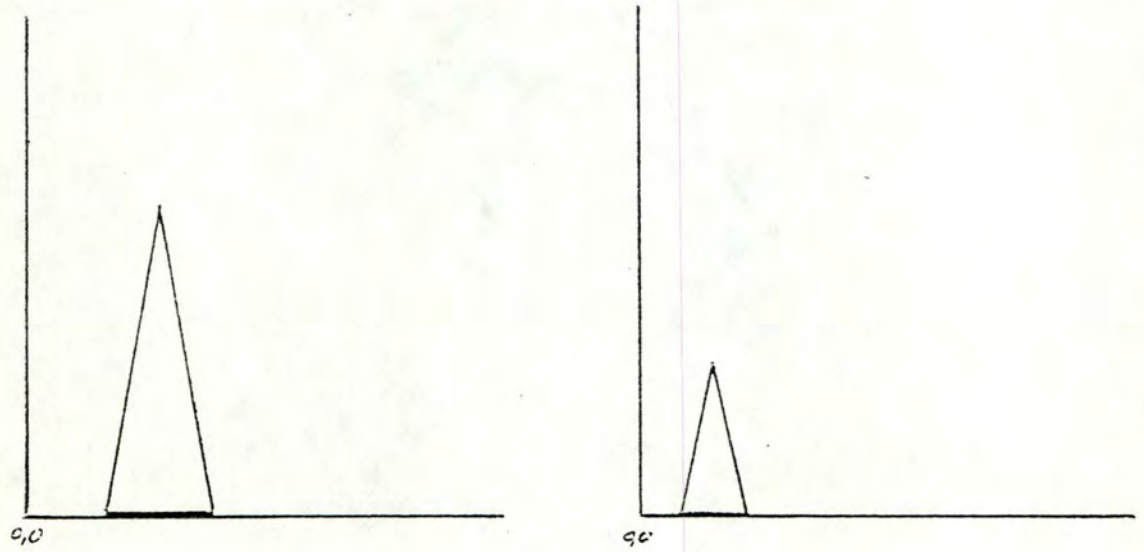
$$x' = xSx$$
$$y' = ySy$$

Par exemple, si on désire agrandir un objet deux fois par rapport à sa dimension originale, on doit choisir Sx et Sy = 2.

Notons que le changement d'échelle s'effectue par rapport à l'origine et qu'il peut être fait de façon distincte, c'est-à-dire pour Sx = Sy. La manière uniforme étant Sx = Sy.
De ce fait, on pourra obtenir dans le cas uniforme où Sx = Sy et que ceux-ci seront 1, une dilatation par rapport à l'origine.

En ce qui concerne la réduction, les valeurs seront comprises entre 0 et 1 et pour ce qui est de la symétrie, c'est-à-dire pour fournir un effet de miroir, on donnera des valeurs négatives.
Ajoutons que l'on obtiendra des distorsions d'images lorsque Sx sera différent de Sy, c'est-à-dire un allongement ou un rétrécissement de celle-ci.

La figure suivante donne un exemple de réduction de 2(Sx = Sy = 2).



Le triangle (20,0), (60,0),(40,100) devient(10,0), (30,0), (20,50).

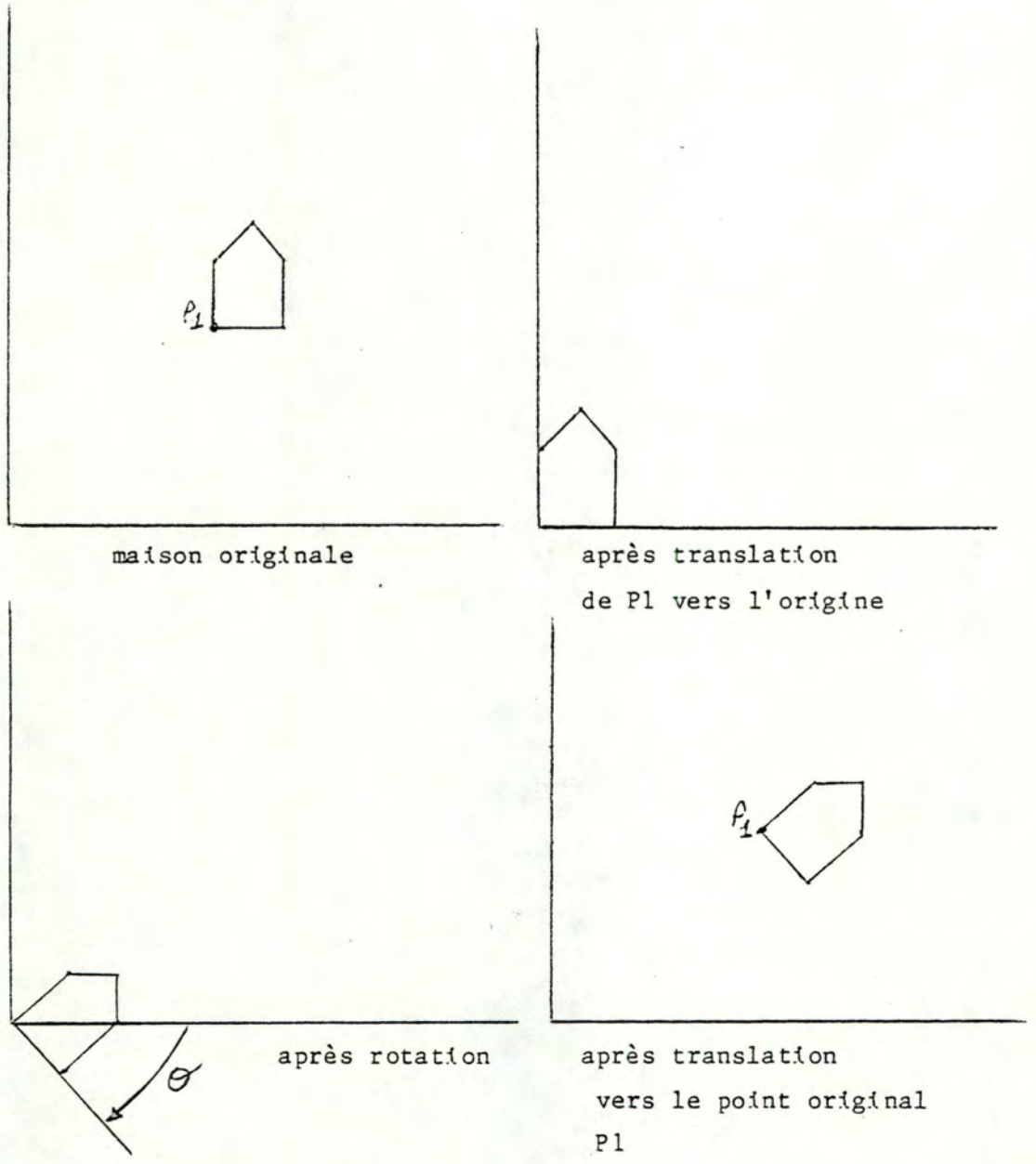
c.2.2. La concaténation

.....

Une séquence de transformation peut être combinée en une seule transformation par un processus de concaténation.
Celui-ci pourra nous servir tout particulièrement lorsque l'on désirera effectuer une rotation ou un changement d'échelle non pas, par rapport à l'origine, mais plutôt en fonction d'un point arbitraire.

Une rotation par rapport à ce type de point, peut être réalisée en effectuant une séquence de trois transformations simples : une translation, une rotation et une translation.

Prenons un exemple : Supposons que l'on désire effectuer une rotation à partir du point P1 de la maison représentée par la figure suivante :



c.2.3. La représentation matricielle

Les transformations en deux dimensions peuvent être représentées de manière uniforme par une matrice 3 x 3. La transformation d'un point (x,y) en un nouveau point (x', y') au moyen de toutes séquences de translations, rotations et changements d'échelle est alors représentée comme suit :

$$[\ x'\ y'\ 1\ ] = [\ x\ \ y\ \ 1\ ] \begin{bmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{bmatrix}$$

où la matrice 3 x 3 spécifie complètement la transformation.

L'intérêt d'une telle représentation est que les règles de concaténation de transformations sont plus simples par rapport à celles utilisées par les équations.
Il est donc plus simple d'utiliser des matrices pour exprimer les transformatons.

-Formulation matricielle des transformations.

Les paramètres d'une matrice de transformation (3 x 3) peuvent être arrangés de telle manière que la matrice représente les transformations simples telles que : la translation, la rotation et le changement d'échelle.

Pour la translation, nous aurons :

$$[\ x'\ y'\ 1\ ] = [\ x\ \ y\ \ 1\ ] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 1 \end{bmatrix}$$

Pour la rotation :

$$[\ x'\ y'\ 1\ ] = [\ x\ \ y\ \ 1\ ] \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Pour le changement d'échelle :

$$[\ x'\ y'\ 1\ ] = [\ x\ \ y\ \ 1\ ] \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

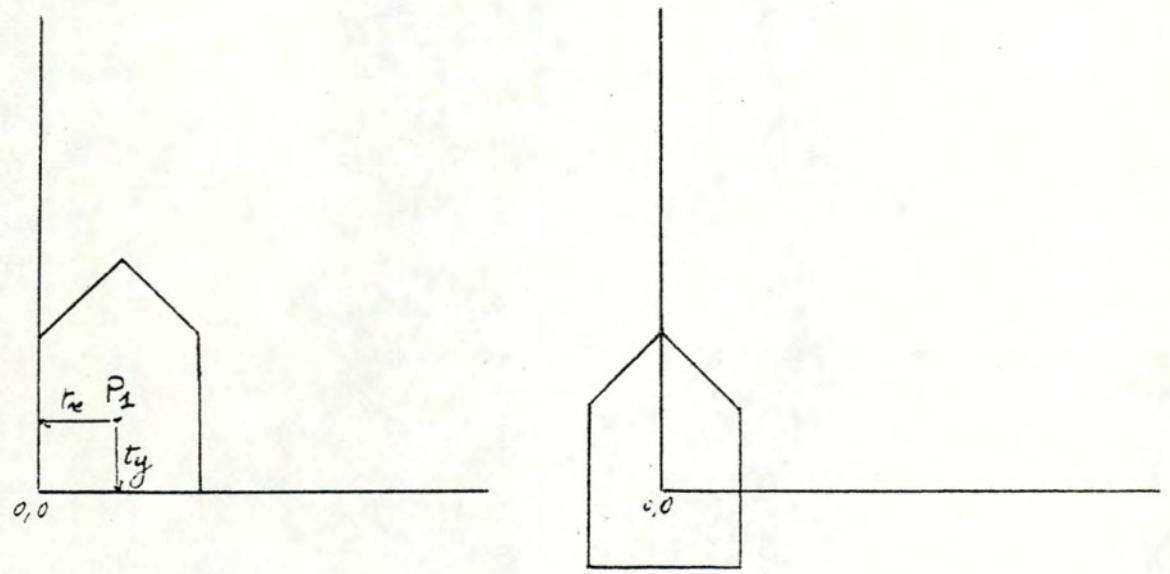
On peut facilement vérifier que les formulations sont équivalentes aux équations qui ont été données précédemment.

-Concaténation de matrices de transformation.

Supposons que l'on désire faire un changement d'échelle, suivi d'une rotation avec P1 comme centre de rotation et de changement d'échelle d'une maison. La séquence de transformations sera la suivante :

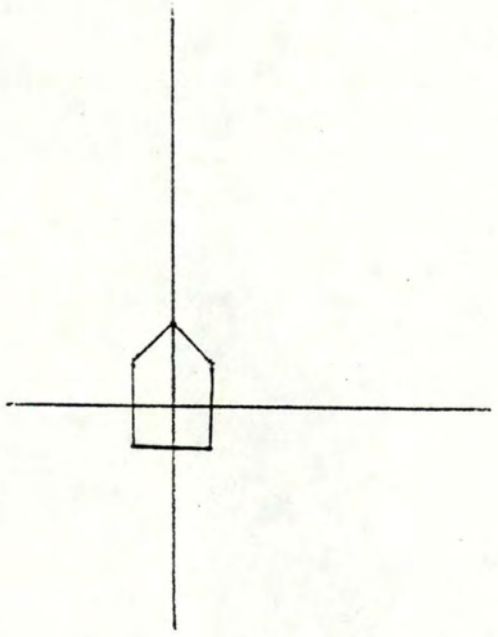
Translation de P1 vers l'origine.
Changement d'échelle.
Rotation.
Translation de l'origine vers P1.

La séquence est représentée par la figure suivante :

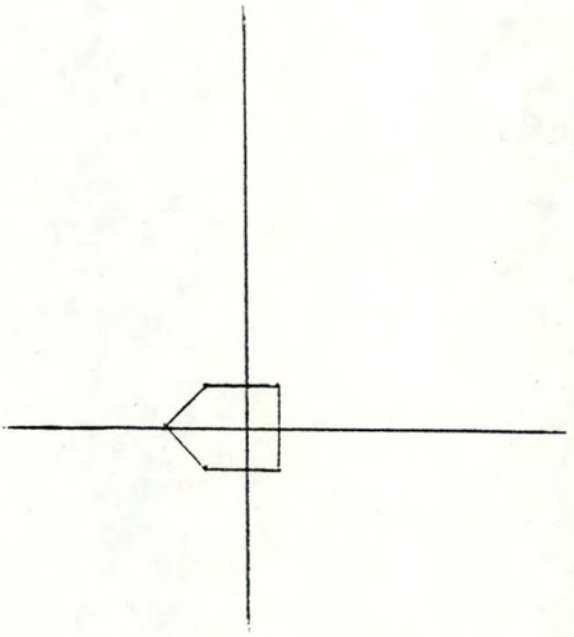


maison

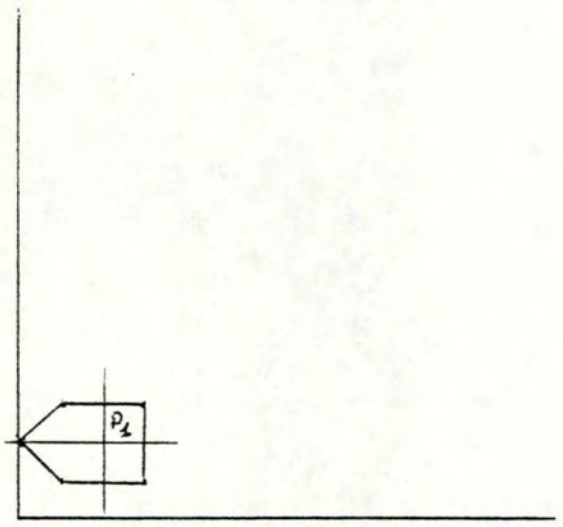
translation de P1, vers l'origine.



changement d'échelle = 1/2



rotation = -90 degrés



translation de l'origine vers P1

La formulation matricielle de ces transformations sera la suivante pour

chaque point extrême de la maison :

Pour la translation de P1 vers l'origine

$$[\ x'\ y'\ 1\ ] = [\ x\ \ y\ \ 1\ ] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Tx & -Ty & 1 \end{bmatrix}$$

Pour le changement d'échelle de 1/2

$$[\ x''\ \ y''\ \ 1\ ] = [\ x'\ y'\ 1\ ] \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Pour la rotation de -90 degrés

$$[\ x'''\ \ y'''\ \ 1\ ] = [\ x''\ \ y''\ \ 1\ ] \begin{bmatrix} \cos(-90) & -\sin(-90) & 0 \\ \sin(-90) & \cos(-90) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Pour la translation de l'origine vers P1

$$[\ x''''\ \ y''''\ \ 1\ ] = [\ x'''\ y'''\ 1\ ] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 1 \end{bmatrix}$$

On peut les concaténer de la façon suivante :

$$[\, x'' \quad y'' \quad 1 \,] = [\, x \quad y \quad 1 \,]$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Tx & -Ty & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-90) & -\sin(-90) & 0 \\ \sin(-90) & -\cos(-90) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 0 \end{bmatrix}$$

Si Tx et Ty sont connus, on peut multiplier les matrices pour obtenir ainsi une seule matrice de transformation et appliquer celle-ci à chaque point extrême de la maison.
Rappelons que ce produit n'est pas commutatif et que seulement dans quelques cas, cette condition n'a aucune importance.

Ces cas sont les suivants :

soit : M1 * M2 M1 et M2 étant des matrices de transformation.

si M1 est	et si M2 est
une translation	une translation
un changement d'échelle	un changement d'échelle
une rotation	une rotation
un changement d'échelle	un changement d'échelle

où le produit est commutatif.

d) conclusion

L'intérêt de la représentation intermédiaire est surtout : qu'elle permet d'effectuer des changements de manière localisée, c'est-à-dire l'insertion d'une opération transférable dans un produit de composition qui représente un gain de temps, par rapport à une solution qui demanderait une reconstruction complète de ce produit (solution d'effacement et de réexécution complète de la partie de dessin que l'on désire modifier à partir du texte).

Une telle situation se présentera lors de l'exécution d'une commande de type opération, en sous-mode graphique.

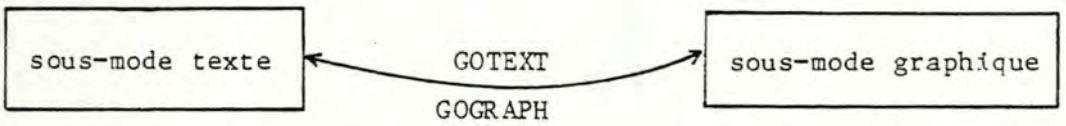
Mais, cette représentation intermédiaire prendra une place supplémentaire en mémoire car, pour chaque opérateur de base, on retiendra ses paramètres et les modificateurs à y appliquer sous la forme de valeurs pour les non-transférables et transférables, afin de pouvoir construire à partir de ces données le produit de composition.

Rappelons que comme tous les textes concernant les opérations sont en mémoire, la représentation intermédiaire prendra donc une place supplémentaire lors de l'exécution de l'un d'eux.

Enfin, on pourrait envisager au niveau des enregistrements concernant les opérations transférables de la représentaton intermédiaire, de stocker chaque opération sous forme matricielle.

4.5.3. LE PASSAGE D'UN SOUS-MODE A L'AUTRE A L'INTERIEUR DU MODE EDITION-OPERATEUR

Après avoir développé chaque mode séparément, nous allons parler du passage du mode texte au mode graphique et inversément et ceci via respectivement les commandes GOGRAPH et GOTEXT.



passage

figure 6

Le passage peut donc s'effectuer dans les deux sens et correspond à un changement d'appui car nous avions vu à la figure 1 du point 4.1. que le sous-mode texte agissait uniquement sur le texte, mais que le sous-mode graphique, en plus du texte nécessite un appui graphique.

Nous allons donc maintenant développer ces deux aspects du passage d'un sous-mode à l'autre.
Bien évidemment, lors du passage d'un sous-mode à l'autre, on reste en édition du même opérateur.

4.5.3.1. LE PASSAGE DU SOUS-MODE TEXTE AU SOUS-MODE GRAPHIQUE.

Le passage du sous-mode texte vers le sous-mode graphique constitue une autre manière d'entrer dans le sous-mode graphique.
Nous avons expliqué une première manière, qui était la commande EDITGRAPH dont l'analyse a été faite du point 4.5.2.1.

L'autre manière est réalisée à l'aide de la commande GOGRAPH. Conceptuellement, cette commande va permettre à l'utilisateur de changer d'appui, de passer d'un appui texte à un appui graphique. Au niveau des principes de réalisation, le texte sera nécessaire, ce dont l'utilisateur n'est pas directement conscient, car il ne dispose à l'écran que de la représentation graphique du dessin support de l'édition.

L'analyse de cette entrée sera similaire à celle faite pour l'entrée via EDITGRAPH.

Cependant quelques nuances sont à apporter.
Que l'on se trouve implicitement dans le cas d'une création ou d'une modification, le texte de l'opérateur en cours d'édition existe, ce qui n'était pas toujours vrai dans l'autre cas. Nous nous retrouverons donc, dans la deuxième situation de point 4.5.2.1.
L'analyse approfondie du passage entre ces deux sous-modes sera donc similaire a celle faite dans la deuxième situation du point 4.5.2.1. et comprendra les quatre mêmes points.
De plus, au niveau de l'exécution, en se basant sur la remarque énoncée en annexe G au point I.1.4, on peut être moins contraignant au niveau erreur de syntaxe.
Toujours au niveau exécution, si on adopte la solution d'une représentation intermédiaire (cfr 4.5.2.3.3.) afin de faciliter l'agissement des commandes du sous-mode graphique, on devra la créer également à ce moment.

4.5.3.2. LE PASSAGE DU SOUS-MODE GRAPHIQUE AU SOUS-MODE TEXTE.

Le passage du sous-mode graphique vers le sous-mode texte constitue une autre manière d'entrer dans le sous-mode texte.
Nous avons expliqué une première manière, qui était la commande EDITEXT dont on a parlé au point 4.5.1.

L'autre manière est réalisée à l'aide de la commande GOTEXT.
Le passage sera immédiat et nettement plus simple que le passage du sous-mode texte au sous-mode graphique. Il suffira d'afficher à l'écran le texte de l'opérateur en cours d'édition.

4.6. LE STOCKAGE D'UN DESSIN EN MODE FICHIER-DESSIN.

Après avoir abordé le mode-édition opérateur, nous avons pensé qu'il était utile d'aborder un point concernant le mode fichier-dessin.

Les deux éléments principaux de ce mode sont :

1. l'exécution d'un opérateur afin d'obtenir le dessin désiré

2. le stockage du dessin

Le premier élément a été longuement abordé au point 4.4.

Le deuxième étant inexistant en mode édition-opérateur, pourrait cependant faire l'objet ici, d'une étude générale.

Il est important de d'abord se rendre compte que, dans ce mode, contrairement à ce qui se passe dans le mode édition-opérateur, le dessin une fois obtenu ne peut subir aucune modification. Il n'est donc pas nécessaire de passer par une représentation intermédiaire permettant de faciliter les modificateurs.
Le stockage du dessin peut être envisagé, entre autre, des deux manières suivantes.
On peut envisager de stocker le dessin sous forme d'une matrice de points, qu'il suffira d'afficher lorsqu'on désirera, par la suite, obtenir ce dessin.
Une mémorisation du dessin sous cette forme, nécessite l'existence d'une limitation de l'espace dessin; afin de connaître la taille de la matrice nécessaire.
Un élément de la matrice est défini comme étant un point de l'écran (pixel). La définition de l'écran étant très précise, une telle matrice sera cependant volumineuse.

Une autre serait de mémoriser le dessin sous forme d'un texte.

Prenons un exemple :

L'opérateur de base "rectangle" possède trois paramètres, qui sont : 2 points et une hauteur.

A cet opérateur est appliqué une série d'opérations déterminées par le ruissellement des modificateurs.
Il est donc nécessaire de traduire ces trois paramètres en, par exemple, 4 points repères définissant les 4 coins du rectangle.
Ensuite, on applique à chacun de ces points, les opérations par les méthodes expliquées au point 4.5.2.3.3., 3ème point; afin de déterminer les 4 points repères résultats.
Les 4 points ainsi trouvés constituent les coordonnées finales, coordonnées des 4 points du rectangle. Il suffit alors de tracer un segment de droite entre ces 4 points pour obtenir la représentation graphique désirée.

Il suffirait donc alors de mémoriser le dessin sous forme d'un texte contenant

- la valeur de l'état écran pour ce dessin

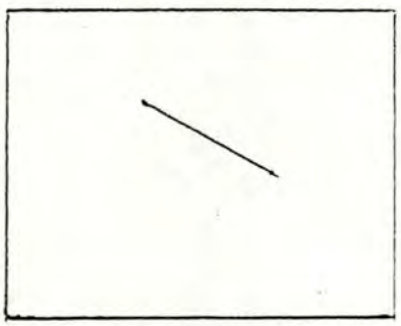
- pour chaque opérateur de base, l'appel à des routines permettant de tracer la représentation graphique désirée à partir des points repères résultats

Nous n'avons pas fait une analyse de ces routines, car il existe de nombreux algorithmes classiques dans la littérature graphique.
Le choix entre ces deux solutions va dépendre d'une caractéristique de la représentation graphique.

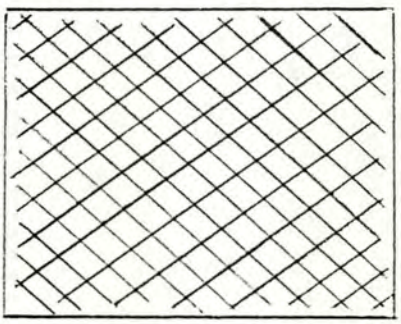
Prenons l'exemple suivant :

soit deux dessins dont les représentations graphiques sont :

A écran



B écran



Dans le cas A, un stockage sous forme de matrice va nous faire perdre énormément de temps d'affichage, vu que celui-ci se fera ligne-ligne.
De plus, la place occupée est importante, pour une quantité limitée d'informations, car dans cette solution, quelque soit la quantité d'informations utiles, le nombre de points stockés est toujours le même.

Dans ce cas, la deuxième solution aurait été plus appropriée, la place nécessaire étant moindre et l'affichage quasi immédiat.

Par contre, dans le cas B, le stockage sous forme de matrice serait plus approprié, car le nombre de points par ligne d'affichage étant assez élévé, on gagnerait du temps par rapport à l'autre solution plus coûteuse au niveau du temps d'exécution des routines, vu leur nombre.

En conclusion, comme nous travaillons sur micro-ordinateur et que nous sommes confrontés à des problèmes de rapidité d'affichage et de place mémoire, on peut dire qu'il faut faire un compromis entre ces deux aspects afin de réaliser un choix au niveau du stockage.

4.7. CONCLUSION.

Nous avons donc, dans ce chapitre consacré aux principes de réalisation, développé plus particulièrement le fonctionnement du mode édition-opérateur, mode central de notre éditeur.
Par manque de temps,nous nous sommes limités au développement de ce point, et des problèmes qui lui sont liés, car seul ce mode constituait la particularité de notre approche.

Les modes fichier-opérateur, aide et initialisation rencontreront les problèmes classiques. Voilà la raison pour laquelle nous ne les avons pas approfondis dans cette quatrième partie.

Reste le mode fichier-dessin.
Les problèmes rencontrés dans ce mode ont été évoqués au niveau du mode édition opérateur : l'exécution d'un opérateur, à laquelle est liée le problème de la syntaxe, la mémorisation du dessin obtenu (ce point a fait l'objet d'une analyse particulière).

Tout au long de cette quatrième partie, nous avons analysé des problèmes, fait des choix et proposé des solutions.
Vu que nous n'avons pas effectué d'implémentation, ce chapitre ne constitue qu'une analyse préliminaire, fournissant des indications pour une réalisation ultérieure, les décisions définitives étant laissées au choix des personnes s'occupant du développement final.

PARTIE 5 : CONCLUSION

(Ladam J-P.)

Le but de ce mémoire étant de proposer une étude d'un éditeur graphique, nous nous sommes donc efforcés de développer une approche personnelle de ce type d'édition.

Rappelons que cette approche repose sur les idées suivantes :

- on ne peut créer un dessin, qu'en passant par la création de la famille à laquelle appartient celui-ci. Ceci s'effectue à l'aide de deux supports, l'un étant le texte de l'opérateur décrivant cette famille, l'autre la représentation graphique d'un élément de cette famille, c'est-à-dire un dessin.

- ajoutons, que le texte de l'opérateur ainsi créé, sera écrit en langage LG, langage que nous avons créé.

- on pourra modifier des caratéristiques particulières liées à une famille, sans changer la définition de celle-ci, lorsqu'elle entre dans la définition d'une autre. Ceci sera possible grâce au concept tout à fait particulier que nous avons introduit : les modificateurs.

A partir de ces éléments, nous avons proposé des principes de réalisation décrivant la manière dont nous envisagions l'exécution d'un opérateur et l'édition de celui-ci.

Nous sommes cependant conscients que ce mémoire présente certaines lacunes dues au fait que nous avons perdu trop de temps au niveau de l'analyse fonctionnelle.

Par conséquent, au niveau des principes de réalisation, certains problèmes réels n'ont pas été abordés, ne l'ont été qu'en partie ou n'ont été que mentionnés, car par manque de temps, nous avons fait appel à des hypothèses simplificatrices.

De plus, le choix que nous avons fait au niveau des commandes de l'éditeur peut être sujet à discussion et l'ensemble des commandes minimales fournies, seront de ce fait sujet à améliorations car, il nous est apparu important de consacrer plus de temps au développement des concepts dont nous désirions que l'approche soit personnelle et créative, ainsi qu'au développement de l'édition d'un opérateur dont l'approche que

l'on proposait était particulière.

Au niveau de l'analyse, différentes améliorations peuvent être apportées à nos concepts et à notre langage.

Nous allons ici les mentionner et relever les implications qu'elles pourraient avoir sur ce que nous avons développé.

D'abord, on remarquera que notre langage ne possède ni structure itérative ou conditionnelle, ni la récursivité.

Qu'adviendra-t-il de nos concepts si on introduisait de telles notions?

En présence de structures de ce type, la définition du concept de famille tel que nous l'avons défini jusqu'à présent, doit subir certains changements: la structure de la famille et la structure de ses dessins n'étant plus identiques. Des problèmes apparaîtront alors également au niveau de la dénomination des noeuds.

Nous aurions pu proposer dans l'analyse fonctionnelle, la possibilité de supprimer un dessin correspondant à un noeud de niveau supérieur à 2, dans la structure d'un opérateur, ce qui aurait pu être effectué à l'aide d'un modificateur local.

Une telle possibilité n'aurait impliqué aucun problème au niveau des structures. Il aurait alors été nécessaire de définir une opération "supprimer" qui dans l'algorithme des modificateurs aurait été considérée, comme une opération non transférable.

Il aurait également été intéressant de permettre à l'utilisateur, de colorier l'intérieur d'un dessin ou d'introduire du texte dans celui-ci.

En ce qui concerne les modificateurs, on peut dire qu'au point de vue méthodologique, cet outil ne force pas l'utilisateur à penser à tout, avant de définir un opérateur. Il est un fait, que l'on pourrait avec plus de réflexion dans certains cas, exprimer ces modificateurs par l'intermédiaire de paramètres. Cependant, on remarquera qu'un tel outil est intéressant par le fait qu'il permet une grande liberté d'action sur un constituant, sans modifier la définition de l'opérateur correspondant à celui-ci.

Enfin, ce mémoire nous a fait découvrir les problèmes engendrés par un travail de longue haleine, effectué en équipe par des personnes ne se connaissant pas au départ, tant du point de vue de la façon de travailler que de celle de penser et d'agir.

LEXIQUE

Composant (Opérateur) : est équivalent à celui d'une famille.

Composé (Opérateur) : est équivalent à celui d'une famille.

Constituant (famille) : un noeud (sauf la racine de la structure) d'une famille représentant :

- soit tous les éléments d'une famille
- soit un sous-ensemble (non-vide) deces éléments.

Composant (famille) : chaque noeud de la sous-structure d'une famille, exceptée la racine.

Composé (famille) : la racine de la structure d'une famille.

Constituant (opérateur) : est équivalent à celui d'une famille.

Dessin : un dessin est constitué d'un ensemble de dessins qui sont soit de base, soit composés.

dessin :: = dessin*/dessin de base
dessin de base :: = droite, cercle...

Dessin composé : c'est un dessin qui peut être décomposé en un certain nombre de dessins, c'est également le dessin associé à la racine de la structure d'un dessin.

Dessin composant : un dessin associé à un noeud de la sous-structure d'un dessin,excepté la racine.

Dessin constituant : un dessin associé à un noeud de la structure d'un dessin,excepté la racine.

Dessin de base : un dessin de base est un dessin indécomposable.

Famille de base : contient des dessins de base ayant même structure.

Famille de dessins : une famille de dessins regroupe des dessins ayant même structure (définissant ainsi les caractéristiques communes aux dessins de la famille).

Famille non de base : contient des dessins composés ayant même structure.

Instruction d'état : dans le langage LG, elle décrit une manière particulière de l'état auquel est associé un identificateur.

Les trois états possible sont :

-COULEUR
-TRAIT
-ECRAN

Instruction d'affectation: cette instruction spécifie qu'une valeur calculée doit être affectée à une variable.

Instruction opérateur : une instruction opérateur permet de décrire la manière dont on s'est servi d'un opérateur ainsi que les opérations que l'on appliquera aux noeuds de la structure appartenant au sous-arbre dont la racine correspond à cette instruction.

liste de qualificateurs : dans le langage LG, cette liste désigne une instruction opérateur $\notin$ au bloc contenant l'instruction opérateur dans laquelle la liste se trouve.

Au niveau de la structure d'un opérateur, elle désigne un constituant de la structure.

Modificateur : permet d'exprimer des opérations sur un constituant quelconque de la structure d'un opérateur; permet d'ajouter à un opérateur une caractéristique non présente dans la définition de celui-ci sans devoir la modifier.

Modificateur local : permet d'exprimer des opérations portant sur un constituant de niveau 2.

Modificateur global : permet d'exprimer des opérations portant sur un composant.

occurence d'opération : pour une opération donnée, une occurence de cette opération sera constituée :
- du nom de l'opération
- des valeurs de paramètres de cette opération.

Opérateur : un opérateur décrit une famille de dessins.

Opérateur composé : décrit une famille non de base. Ce sont des opérateurs définis par l'utilisateur à l'aide d'opérateurs de base et (ou d'opérateurs composés créés par lui précédemment. Ils sont écrits en langage LG.).

Opérateur de base : décrit une famille de base. Ce sont des opérateurs prédéfinis. Ils constituent les matériaux de base pour l'utilisateur.

Opération non transférable : couleur, trait.

Opération transférable : rotation, translation, symétrie, réduction, dilatation.

Sous-structure d'une famille : les deux premiers niveaux de la structure d'une famille.

Sous-structure d'un opérateur : est équivalente à celle d'une famille

Sous-structure d'un dessin : la sous-structure d'un dessin est composée des deux premiers niveaux de la structure du dessin.

structure d'une famille : elle sera correspondante à celle des des dessins de la famille, les noeuds correspondant un à un.

Structure d'un opérateur : est équivalente à celle d'une famille.

Structure d'un dessin composé : un dessin composé possède une structure arborescente, chaque noeud de cette structure étant un dessin.

BIBLIOGRAPHIE

(AL82) Apple LOGO reference Manual
Logo computer systems Inc 1982

(BE82) BEQUET Henri (mémoire U.L.B.)
Graphsoft : un logiciel de saisie graphique 1982 - 1983

(CD82) Computer Design July 1982
Special report on graphics system design Technology
Working Toward Standards in Graphics
by Fred E. Langhorst p. 177 to p. 182

(DL81) DEBACKER Serge, LESTRADE Michel
Etude et réalisation d'un éditeur de texte.
1980-81.

(FO82) J. FOLEY, A. VANDAM
"Fundamentals of interactive computer graphics"
Addison - Wesley, Systems programming series 1982

(GK82) "Graphical Kernel System : functional description"
Draft international Standard ISO/DIS January 1982

(HO82) F. HOPGOOD
"The road to graphics standards"
Computer - aided Design Vol 14 no 4 July 1982

(MI78) J. MICHENER, A. VANDAM
"A functional overview of the core system with
glossary" Computing Surveys Volume 10, no 4 December 1978

(NS79) W. M. NEWMAN, R.F. SPROULL
"Principles of Interactive Computer Graphics"
(second edition)
Mc Graw-Hill Book Company 1979

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX - NAMUR (BELGIQUE),

INSTITUT D'INFORMATIQUE

PROPOSITION ET ANALYSE D'UN EDITEUR GRAPHIQUE :

Annexes

Jean-Paul LADAM

Jacques LEFEBVRE

Promoteur : Cl. Cherton

Mémoire présenté en vue
de l'obtention du grade de
LICENCIE ET MAITRE EN INFORMATIQUE

ANNEE ACADEMIQUE 1983 - 1984

ANNEXES

Annexe A : Tables.
Annexe B : Syntaxe BNF du langage LG.
Annexe C : Diagrammes syntaxiques du langage LG.
Annexe D : Les opérateurs de base.
Annexe E : Les opérations.
Annexe F : Les états.
Annexe G : Les commandes de l'éditeur.
Annexe H : Le ruissellement des modificateurs:

- algorithme
- exemple

(Annexes A, B, C, D, E, F, G : LADAM J.-P.
LEFEBVRE J.

Annexe H : LADAM J.-P.)

ANNEXE A : TABLES

TABLES

I. Table des identificateurs standarts.

OPERATEURS:
- droite
- rectangle
- cercle
- arc
- polygone régulier
- triangle
- élipse
- parallélogramme
- ligne polygonale

ETATS :
- couleur
- trait
- écran

OPERATIONS :
- rotation
- translation
- symétrie - pt
- symétrie - dr
- réduction
- dilatation
- couleur
- trait

II. Table des mots réservés

- opérateur

ANNEXE B : SYNTAXE BNF DU LANGAGE LG

Backus - Naur Form BNF

Nota : Les symboles suivants sont les méta-symboles du formalisme BNF et non pas des symboles du langage graphique.

:: = | { }

Les accolades indiquent une éventuelle répétition des symboles qu'elles englobent, zéro, une ou plusieurs fois
En général A :: = {B} est une forme abrégée de

A :: = <vide> | AB

BNF

<opérateur> :: = <en-tête d'opérateur> <bloc>.

<en-tête d'opérateur> :: =
opérateur <identificateur d'opérateur> <liste de paramètres formels> ;

<identificateur d'opérateur> :: = <identificateur>

<identificateur> :: = <lettre> { <lettre ou chiffre> }

<lettre ou chiffre> :: = <lettre> | <chiffre>

<liste de paramètres formels> :: = (<identificateur de paramètres formels> { , <identificateur de paramètres formels> })

| <vide>

<identificateur de paramètres formels> :: = <identificateur>

<bloc> :: = <instruction> { ; <instruction> }

<instruction> :: = <instruction opérateur> | <instruction d'affectation> | <instruction d'état>

<instruction d'opérateur> :: = <qualificateur d'opérateur> : <identificateur d'opérateur>

<modificateur> <liste de paramètres effectifs>

<qualificateur d'opérateur> :: = <identificateur>

<liste de paramètres effectifs> :: = (<paramètre effectif> { , <paramètre effectif> }) | <vide>

<paramètre effectif> :: = <paramètre effectif valeur> | <paramètre effectif variable> | <paramètre effectif expression>

<paramètre effectif variable> :: = <variable>

<paramètre effectif valeur> :: = <nombre>

<réel non signé> :: = <chiffre> { <chiffre> } . <chiffre> { <chiffre> } | <chiffre> { <chiffre> }

<nombre> :: = <nombre non signé> | <signe> <nombre non signé>

<nombre non signé> :: = <réel non signé>

<paramètre effectif expression> :: = <expression>

<instruction d'état> :: = <identificateur d'état> <liste de paramètres effectifs>

<identificateur d'état> :: = <identificateur dans table état>

<modificateur> :: = <liste de modificateurs globaux> <liste de modificateurs locaux>

<liste de modificateurs globaux> :: = [<modificateur global> { , <modificateur global> }]
| <vide>

<liste de modificateurs locaux> :: = [<modificateur local> { , <modificateur local> }]
| <vide>

<modificateur global> :: = <opération>

<opération> :: = <identificateur d'opération> <liste paramètres effectifs>

<identificateur d'opération> :: = <identificateur dans la table d'opération>

<modificateur local> :: = <liste de qualificateur d'opérateur> : <opération> { , <opération> }

<liste de qualificateur d'opérateur> :: = <qualificateur d'opérateur> { - <qualificateur d'opérateur> }

<instruction d'affectation> :: = <variable> : = <expression>

<variable> :: = <identificateur>

<expression> :: = <terme> | <expression> <opérateur additif d'expression> <terme>
| <signe> <terme>

<terme> :: = <facteur> | <terme> <opérateur multiplicatif d'expression> <facteur>

⟨facteur⟩ :: = ⟨variable⟩ | ⟨nombre non signé⟩ | (⟨expression⟩)

⟨vide⟩ :: =

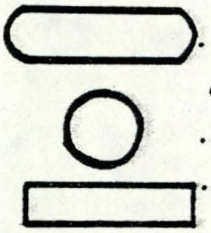
⟨signe⟩ :: = + | -

⟨opérateur additif d'expression⟩ :: = + | -

⟨opérateur multiplicatif d'expression⟩ :: = * | /

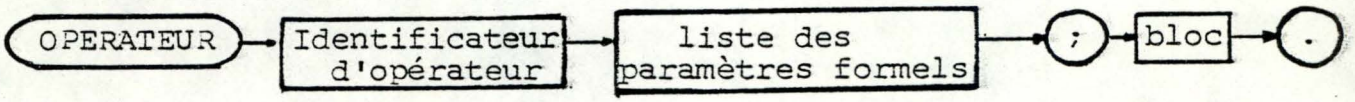
ANNEXE C : DIAGRAMMES SYNTAXIQUES DU LANGAGE LG

Diagrammes syntaxiques définissant la structure générale d'un opérateur

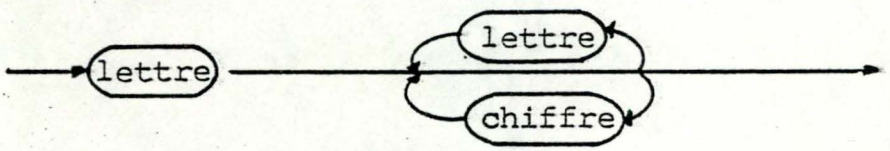


- représente un mot réservé en LG ou des entités syntaxiques qui ne sont pas définies ultérieurement. (lettre, chiffre).
- représente un séparateur du LG
- représente une entité syntaxique qui est définie par un centre diagramme syntaxique.

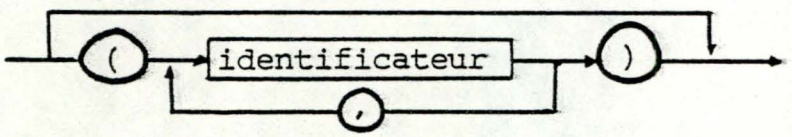
Opérateur



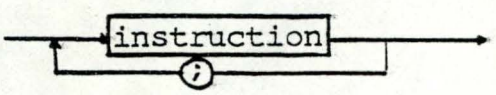
Identificateur



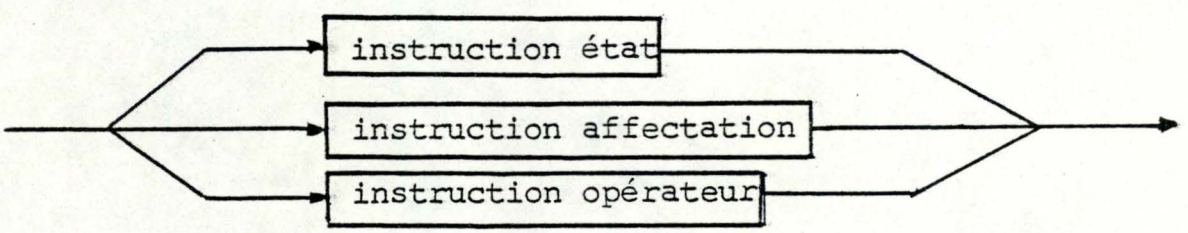
Liste de paramètres formels



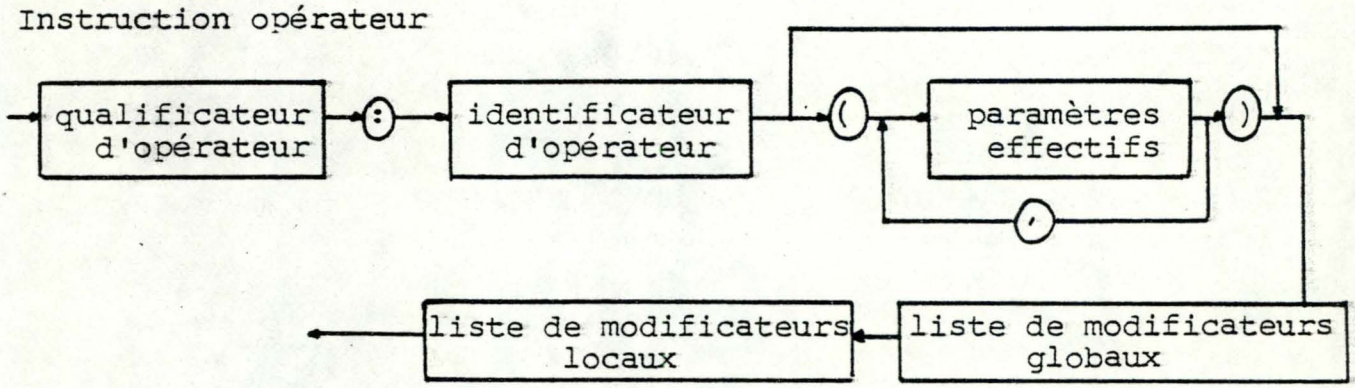
Bloc



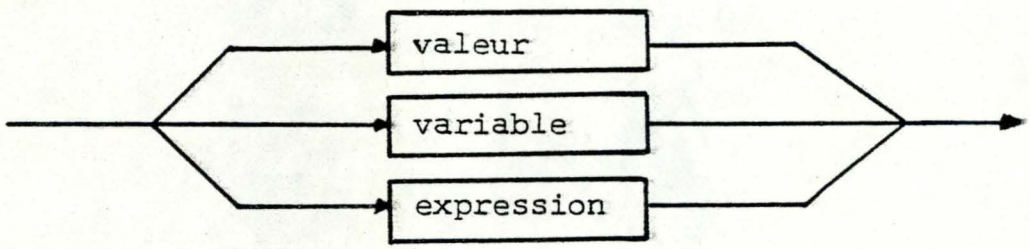
Instruction



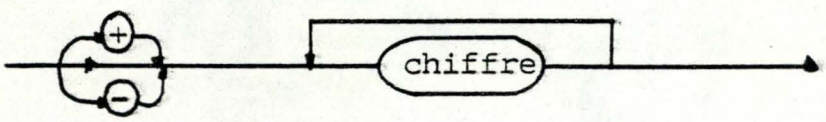
Instruction opérateur



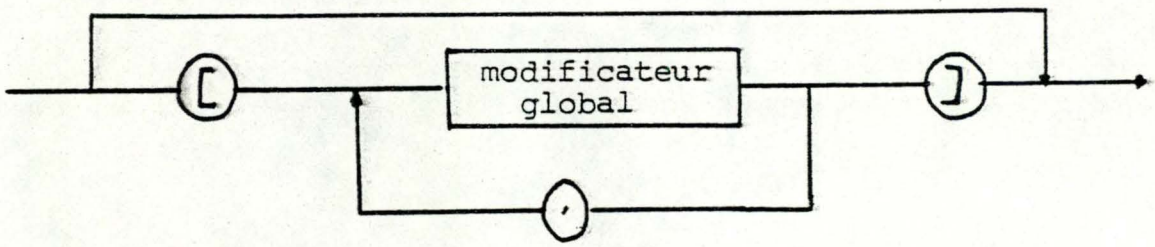
Paramètres effectifs



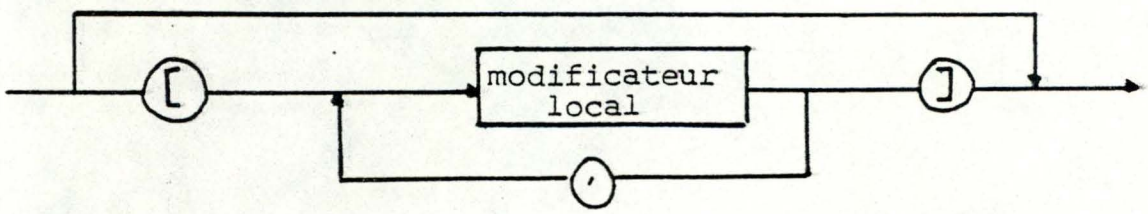
Valeur



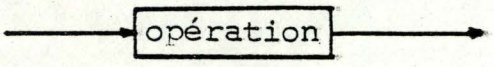
Liste de modificateurs globaux



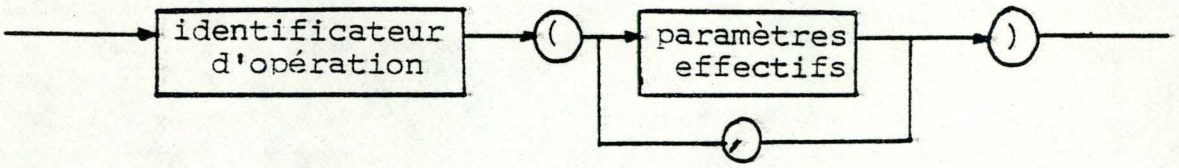
Liste de modificateurs locaux



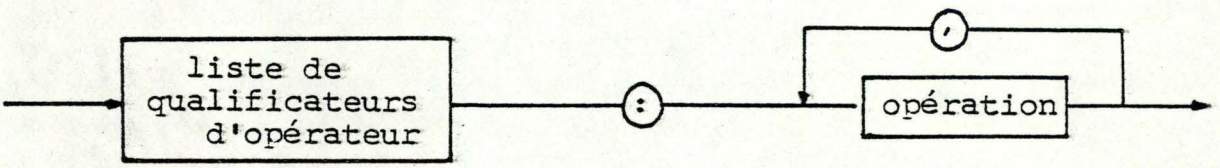
Modificateur global



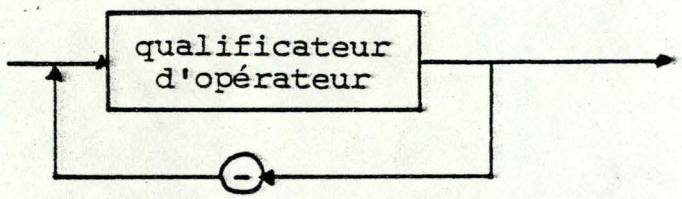
Opération



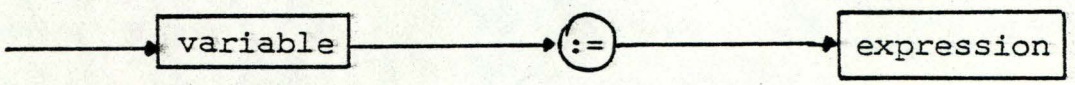
Modificateur local



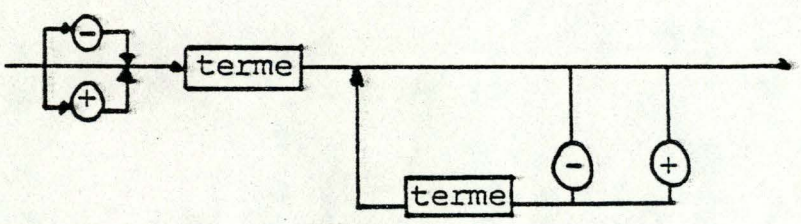
Liste de qualificateurs d'opérateur



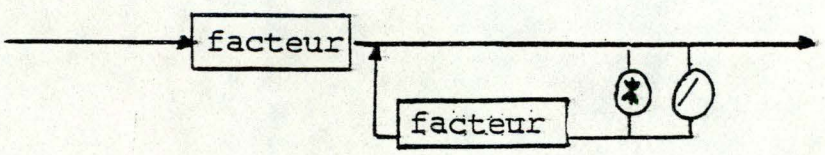
Instruction affectation



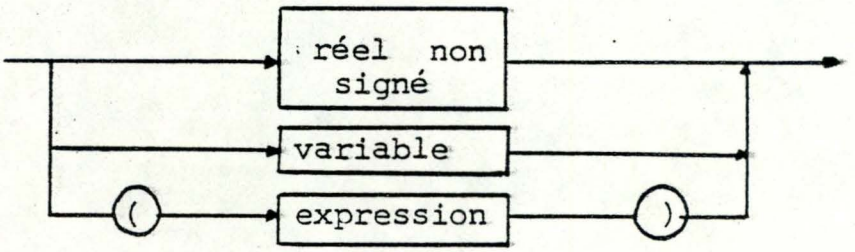
Expression



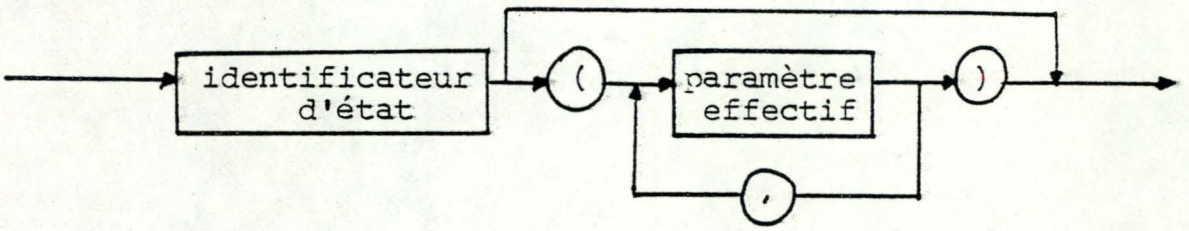
Terme



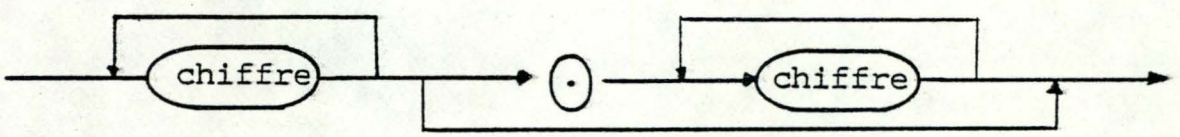
Facteur



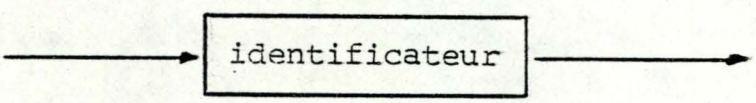
Instruction d'état



Réel non signé



Qualificateur, variable



ANNEXE D : Les opérateurs de base.

Précisons que par convention pt $_1$, pt $_2$, pt $_3$, pt utilisés dans les listes de paramètres, des opérateurs de base seront des coordonnées constituées chacune de 2 nombres.
La définition de nombre se trouve dans la BNF du langage à l'annexe B.

DROITE (pt_1 , pt_2 , type de prolongement).

L'opérateur de base DROITE trace la droite définie par le segment joignant le point " pt_1 " et le point " pt_2 " en tenant compte de la valeur du type de prolongement.

Notion de prolongement :

Il y a 3 prolongements possibles :

- A : du point " pt_1 " jusqu'au bord de l'écran le plus proche.
- B : du point " pt_2 " jusqu'au bord de l'écran le plus proche.
- C : A et B

"type de prolongement" :

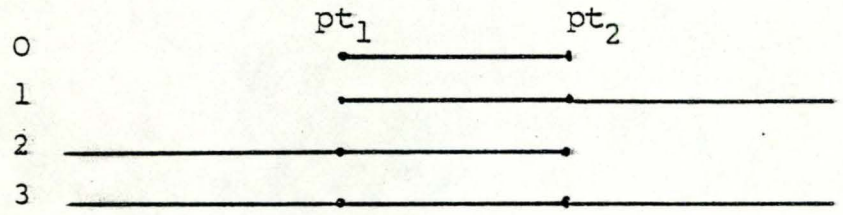
Il y a 4 valeurs possibles :

0 si pas de prolongement

1 si A

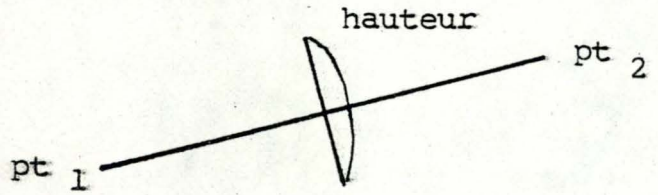
2 si B

3 si C



RECTANGLE (pt_1 , pt_2 , hauteur)

L'opérateur RECTANGLE permet de tracer un rectangle dont la droite définie par "pt_1" et "pt_2" est une médiane et dont "hauteur" est la longueur de l'autre médiane.



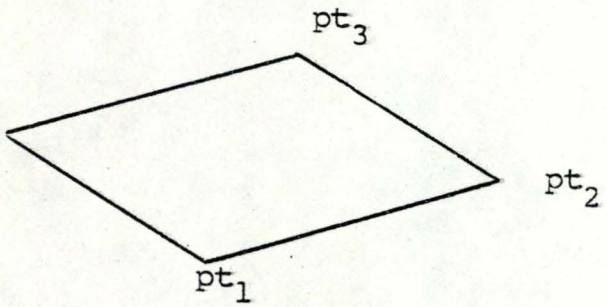
avec "hauteur" = 1 nombre.

PARALLELOGRAMME (pt_1 , pt_2 , pt_3)

L'opérateur de base PARALLELOGRAMME permet de tracer un parallélogramme passant par les points "pt_1" "pt_2" "pt_3" en respectant la règle suivante :

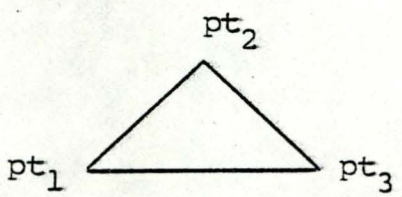
"pt_1", "pt_2", "pt_3" ne peuvent se trouver sur la même ligne.

Contrainte : les 3 points définissent obligatoirement une droite passant par "pt_1" et "pt_2" et une droite passant par "pt_2" et "pt_3".



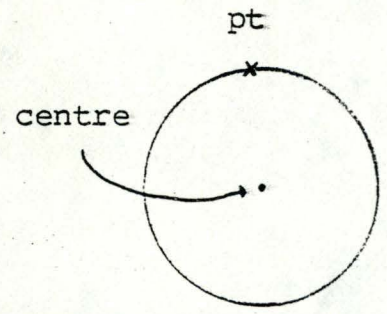
TRIANGLE (pt_1, pt_2, pt_3)

L'opérateur de base TRIANGLE permet de tracer un triangle passant par les points "pt_1", "pt_2", "pt_3".



CERCLE (centre, pt)

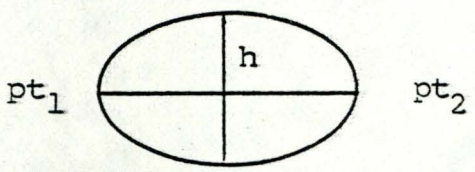
L'opération de base CERCLE permet de tracer un cercle de centre "centre" et passant par le point "pt".



avec "centre" = coordonnées (2 nombres).

ELLIPSE (pt_1, pt_2, hauteur)

L'opérateur de base ELLIPSE permet de tracer une élipse dont la droite définie par "pt_1", "pt_2", est une médiane, dont "hauteur" est la longueur de l'autre médiane.



avec "hauteur" = 1 nombre.

LIGNE POLYGONALE (pt_1, ..., pt_{20}, close

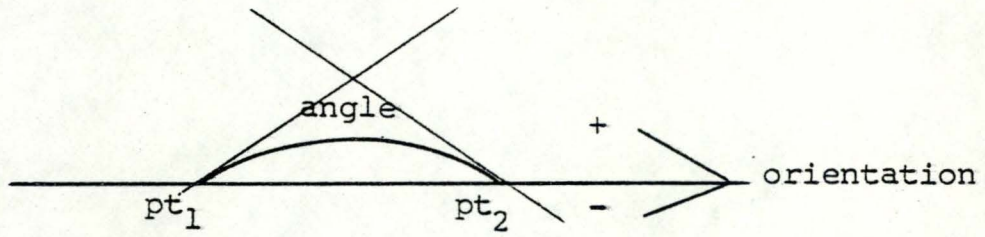
L'opérateur de base LIGNE POLYGONALE permet de tracer une ligne polygonale passant par les points "pt_1", ...pt_{20}" en respectant la règle suivante :

- Les segments successifs de la ligne polygonale sont déterminés par l'ordre des points dans la liste des paramètres.
- La fermeture ou la non-fermeture de la ligne polygonale étant déterminée par le paramètre "close".

"close" prendra la valeur 0 pour oui (fermeture)
1 pour non

ARC (pt_1, pt_2, angle, orient)

L'opérateur de base ARC permet de tracer un arc ayant pour origine le point "pt_1" et arrivée le point "pt_2" avec une orientation "orient" et un angle "angle".

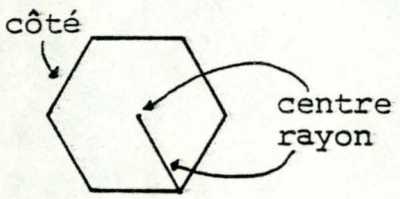


avec $0 \leqslant$ "angle" $\leqslant 180$

"orientation" = + ou -

POLYGONE REGULIER (pt, rayon, nombre côtés)

L'opérateur de base POLYGONE REGULIER permet de tracer un polygone régulier de centre "pt", de rayon "rayon" dont le nombre de côtés est déterminé par "nombre de côtés"



nombre de côtés = 6

avec "rayon" = 1 nombre

"centre" = coordonnées (2 nombres)

ANNEXE E : Les opérations

Précisons que par convention, pt_1, pt_2, pt utilisés dans les listes de paramètres des opérations seront des coordonnées constituées chacune de deux nombres.

La définition de nombre se trouve dans la BNF du langage à l'annexe B.

ROTATION (centre,angle)

L'opération ROTATION permet d'effectuer une rotation de centre "centre" et d'angle "angle".
"angle" pourra prendre une valeur comprise entre 0 et 360.
"centre" est défini comme étant un point (pt).

TRANSLATION (longueur, orientation)

L'opération TRANSLATION permet d'effectuer une translation de longueur "longueur" et d'orientation "orientation" .
"orientation" pourra prendre une valeur comprise entre 0 et 360.
"longueur" sera un nombre.

SYMETRIE - PT (pt)

L'opération SYMETRIE - PT permet d'effectuer une symétrie par rapport à un point "pt".

Définitions : - 2 points sont symétriques par rapport à un 3ème lorsque celui-ci est au milieu de la droite qui joint les 2 points.

- 2 objets sont symétriques par rapport à un point lorsque les points des objets sont 2 à 2 symétriques par rapport à un point.

SYMETRIE - DR (pt_1, pt_2)

L'opération SYMETRIE - DR permet d'effectuer une symétrie par rapport à une droite passant par 2 points définis par les paramètres "pt_1" et "pt_2".

Définition de la symétrie par rapport à une droite :

- 2 points sont symétriques par rapport à une droite, lorsque cette droite est perpendiculaire sur le milieu de la droite qui joint les deux points.
- 2 objets sont symétriques par rapport à une droite, lorsque leurs points sont 2 à 2 symétriques par rapport à la droite.

REDUCTION (Centre, coef-x, coef-y)

L'opération REDUCTION permet d'effectuer une réduction dont le centre est l'origine d'un système d'axes X et Y.
A partir de ce système, il est possible de déterminer une valeur différente de REDUCTION pour l'axe X aussi bien que pour l'axe Y.
Ces valeurs sont définies par les paramètres "coef-x" pour X et "coef-y" pour Y

- centre = un point (pt)
- coef-x = nombre entier >0
- coef-y = nombre entier >0

DILATATION (centre, coef x, coef y)

L'opération DILATATION permet d'effectuer une dilatation dont le centre est déterminé par "centre".
Le centre est l'origine d'un système d'axe (X Y).

Il est possible de déterminer une valeur différente de dilatation pour l'axe X (coef x) que pour l'axe Y (coef y).

- centre = un point (pt)
- coef x = un nombre entier >0
- coef y = un nombre entier >0

COULEUR (type, intensité)

L'opération COULEUR permet de définir la couleur utilisée pour la plume ainsi que l'intensité.
Les couleurs et intensités possibles sont définies par les paramètres "type" et "intensité".
Les différentes valeurs possibles de "type" et "intensité" sont fonctions du système utilisé.

- type = un nombre
- intensité = un nombre

TRAIT (épaisseur, type)

L'opération TRAIT permet de définir le trait utilisé pour la plume. Le trait est caractérisé par une épaisseur, et un type défini par les paramètres "épaisseur" et "type".
Les valeurs possibles de "type" sont: -continu
-pointillé

d'"épaisseur " sont : -épais
-moyen
-fin

ANNEXE F : Les états

Précisons que par convention, Couleur; Intensité, Epaisseur et type utilisés dans les listes de paramètres des états sont des nombres.

COULEUR (type, intensité).

L'état COULEUR permet de définir la couleur utilisée pour la plume ainsi que l'intensité.
Les couleurs et intensités possibles sont définies par les paramètres "type" et "intensité".
Les différentes valeurs possibles de "type" et "intensité" sont fonctions du système utilisé.

type = 1 nombre
intensité = 1 nombre

TRAIT (épaisseur, type)

L'état TRAIT permet de définir le trait utilisé pour la plume.
Le trait est caractérisé par une épaisseur et un type définis par les paramètres "épaisseur" et "type".
Les valeurs possibles de type sont :
- continu
- pointillé

d'épaisseur sont :
- épais
- moyen
- fin

ECRAN (Couleur, Intensité)

L'état ECRAN permet de définir la couleur de fond utilisée pour l'écran ainsi que l'intensité.
Les couleurs et intensités possibles sont définies par les paramètres "couleur" et "intensité".
Les différentes valeurs possibles de "couleur" et "intensité" sont fonction du système utilisé.

Couleur = 1 nombre
Intensité = 1 nombre

ANNEXE G : Les commandes de l'éditeur.

PLAN

Il est nécessaire de se référer à la BNF (annexe B) pour la définition de tout paramètre d'une commande mis entre crochet (< >).

I. Les commandes du mode édition-opérateur.

I.1. Les commandes du sous-mode texte.

I.1.1. (EDT) EDITEXT <nom opérateur>

Avec <nom opérateur> pouvant être omis.

Cette commande permet d'entrer dans le mode texte
Si "nom opérateur" est présent alors
- Si "nom opérateur" correspond au nom d'un opérateur existant dans le système, alors l'édition en sous-mode texte démarrera avec le texte de l'opérateur choisi. Si celui-ci n'est pas présent en mémoire, il sera préalablement chargé.
- Si "nom opérateur" ne correspond pas au nom d'un opérateur existant dans le système, alors l'édition en mode texte démarrera avec uniquement la ligne suivante : "OPERATEUR nom opérateur" qui correspond à une partie de l'en-tête de l'opérateur (voir langage LG).

Si "nom opérateur" est omis, alors l'édition en sous-mode texte concernera le même opérateur que lors de la dernière édition (aussi bien texte que graphique) et elle démarrera avec le texte de l'opérateur en question. Si il n'y avait pas d'édition précédente, alors un message d'erreur est communiqué.

I.1.2. Commandes liées aux mouvements du curseur.

Ces commandes permettent de déplacer le curseur dans un texte.

- ^F bouge le curseur vers la droite d'une position-caractère (1 caractère).

- ^B bouge le curseur vers la gauche d'une position - caractère (1 caractère).
- ^N bouge le curseur vers le bas d'une position - ligne (1 ligne), le curseur va vers le caractère juste en dessous du caractère de la ligne courante. Si la ligne suivante est plus petite, le curseur se positionne à la fin. Si le curseur est à la fin du texte, il ne bouge plus.
- ^P bouge le curseur vers le haut d'une postition - ligne. Le curseur va vers le caractère juste au-dessus du caractère de la ligne courante. Si la ligne précédente est plus petite, le curseur se positionne à la fin. Si le curseur est au début du texte, il ne bouge plus.
- ^A bouge le curseur vers le début de la ligne courante.
- ^E bouge le curseur vers la vin de la ligne courante.
- ESC > bouge le curseur à la fin du texte.
- ESC < bouge le curseur vers le début du texte.

I.1.3. Commandes d'insertion et de suppression.

- RETURN permet de créer une nouvelle ligne.
  Indique que l'utilisateur a terminé une ligne.
  Le curseur va se placer au début de la ligne écran suivante.
- DELETE écrase le caractère à la gauche du curseur. Le curseur revient d'un caractère.
- ^D écrase le caractère à la position du curseur.
- ^K Supprime le texte de la position du curseur jusqu'à la fin de la ligne du texte.
  La ligne est retenue dans une zone de sauvetage.
- ^Y insère une copie de la ligne contenue dans la zone de sauvetage à la position courante du curseur. (Hors de l'éditeur remet la dernière ligne tapée).

I.1.4. Commande de sortie du sous-mode texte.

(GG) GOGRAPH <liste de paramètres effectifs>
permet de passer du sous-mode texte au sous-mode graphique tout en restant en édition du même opérateur. On obtient donc un dessin support de l'édition.
Ce dernier sera obtenu si l'éxécution du texte de l'opérateur que l'on édite s'effectue sans rencontrer des problèmes de syntaxe. Ceci signifie que les opérateurs successivement appelés (chargés si nécessaire) pour réaliser cette exécution soient syntaxiquement corrects. De plus, le nombre d'éléments de la liste des paramètres effectifs doit être égal au nombre de paramètres formels liés à cet opérateur. Si ce n'est pas le cas : un message d'erreur est communiqué.
De plus, à la fin de l'exécution, le système fournira la liste des variables (si cette exécution s'est déroulée sans problèmes).

Remarque : on peut envisager au niveau de l'exécution, d'être moins contraignant concernant les erreurs de syntaxe. On pourrait envisager que l'absence d'instruction opérateur dans le texte de l'opérateur en cours d'édition (et seulement lui) n'empêche pas l'entrée en mode graphique (afin de rendre ce passage plus souple car un opérateur en édition est un opérateur en cours de création ou de modification).

I.1.5. Commande de vérification de syntaxe.

- VS permet d'effectuer une vérification de la syntaxe du texte de l'opérateur que l'on édite.
  Elle communique une liste des erreurs ainsi que l'origine de celles-ci.

I.1.6. Commandes de sortie de l'édition.

- ^C moyen pour sortir du sous-mode texte et de l'édition. On sauve le texte de l'opérateur et ajoute un carriage return à la dernière ligne tapée.
- ^G avorte une édition. Tous les changements sont ignorés La définition de l'opérateur ne change pas. La commande EDT permet de retrouver l'état antérieur au ^G si celle-ci a été tapée par inadvertance.

I.1.7. Commandes de défilement de l'écran.

- ^V passe à la page suivante. Le curseur se positionne au début de la dernière ligne écran qui devient la première. Un écran faisant 22 lignes-écran.
- ESC V passe à la page précédente.
- ^L réaffichage de la même page.

I.1.8. Commande de sauvetage du texte.

- ^S permet de sauver le texte de l'opérateur.

I.1.9. Commande de renseignements sur la structure d'un opérateur.

- ^H <nom opérateur> permet d'obtenir l'arbre de structure de l'opérateur de nom "nom opérateur". Si la structure n'est pas disponible, un message est affiché.

I.2. Les commandes du mode graphique.

I.2.1. (EDG) EDITGRAPH <nom opérateur> <liste de paramètres effectifs>

Avec <nom opérateur> et/ou <liste de paramètres effectifs> pouvant être omis.
Cette commande permet d'entrer dans le sous-mode graphique de l'éditeur afin de créer ou de modifier un opérateur.
Lorsque l'on se trouve en création d'un opérateur, l'écran reste vierge c'est-à-dire qu'il n'existe au départ aucun dessin comme support de l'édition. De plus, le texte de l'opérateur contiendra uniquement la ligne suivante :
OPERATEUR <nom opérateur> ligne qui correspond à l'en-tête de l'opérateur (voir langage LG).
En modification, l'édition démarrera avec à l'écran la représentation graphique du dessin comme support et ceci sous certaines conditions (voir analyse).

Analyse:

Si <nom opérateur> est omis
 alors Si <liste de paramètres effectifs> est omis

 alors l'édition en mode graphique concernera le même opérateur que lors de la dernière édition (aussi bien texte que graphique). On démarrera l'édition avec à l'écran la représentation graphique support de l'édition précédente.

 sinon erreur, on ne peut rentrer en mode graphique.

 sinon Si <nom opérateur> est le nom d'un opérateur existant dans le système et que le nombre d'éléments de la <liste de paramètres effectifs> correspond au nombre de paramètres formels de cet opérateur

 alors exécuter l'opérateur (les textes des opérateurs appelés successivement seront chargé en mémoire s'il ne s'y trouve pas).

Si l'exécution se déroule sans problème (A syntaxique)
alors - On obtient à l'écran la représentation graphique voulue. De plus, on fournit la liste des variables.
sinon - On ne peut entrer dans le mode graphique. On peut fournir des renseignements concernant les erreurs.
sinon Si le problème vient d'une erreur au niveau des paramètres
alors affichage d'un message d'erreur
on ne peut entrer en mode graphique.
sinon - l'édition en mode graphique démarrera avec un écran vierge.
- le texte de l'opérateur contiendra uniquement la ligne suivante :
OPERATEUR nom opérateur
ligne qui correspond à l'en-tête de l'opérateur (voir langage LG).

Remarque :

On peut envisager au niveau de l'exécution d'être moins contraignant concernant les erreurs de syntaxe.
On pourrait envisager que l'absence d'instruction opérateur dans le texte de l'opérateur en cours d'édition (et seulement lui) n'empêche pas l'entrée en mode graphique (afin de rendre le passage plus souple, car un opérateur en édition est un opérateur en cours de création ou de modification).

I.2.2. Sortie de l'édition.

- ^C moyen pour sortir du sous-mode graphique et de l'édition. On sauve le texte de l'opérateur.
- ^G avorte une édition. Tous les changements sont ignorés. La définition de l'opérateur ne change pas. La commande EDG permet de retrouver l'état antérieur au ^G si celle-ci a été tapée par inadvertance.

I.2.3. Sortie du Mode graphique.

(GT) GOTEXT permet de passer du sous-mode graphique au sous-mode texte tout en restant en édition du même opérateur.

I.2.4. Dessin

I.2.4.1. La commande de traçage

(TR) TRACER <nom opérateur> <liste de paramètres effectifs>

: <nom dessin>

Permet d'exécuter l'opérateur désigné par "nom opérateur". On désigne le résultat de l'exécution de l'opérateur par "nom dessin".
Si "<nom dessin>" est omis, le nom de la réalisation de l'opérateur sera généré d'office (nom d'opérateur + n° chiffre).
Une validation sera effectuée sur le "nom opérateur" et sur la"liste des paramètres effectifs". Si le nom opérateur n'∄ dans le système et/ou que le nombre d'éléments de la "liste des paramètres effectifs " ne correspond pas au nombre d'éléments de la liste des paramètres formels de l'opérateur désigné, un message d'erreur sera communiqué.
Tous les textes nécessaires et ne se trouvant pas en mémoire centrale, seront chargés successivement lors du déroulement de l'exécution de l'opérateur désigné par cette commande.

Précisons que le "nom opérateur" désignera 2 types d'opérateurs :

- les opérateurs de base c'est-à-dire :
  - DROITE
  - RECTANGLE
  - CERCLE
  - ARC
  - POLYGONE REGULIER
  - TRIANGLE
  - ELLIPSE
  - PARALLELOGRAMME
  - LIGNE POLYGONALE
- les opérateurs composés créés par l'utilisateur.

En ce qui concerne les paramètres des opérateurs, si ceux-ci représentent des coordonnées, l'utilisateur pourra les désigner de la manière suivante :

(*) {
A pour l'avant dernière marque non identifiée
D pour la dernière marque non identifiée
. pour l'endroit où on est (curseur)
! num pour la marque identifiée par num
x,y par le point dont x et y sont les coordonnées.
}

La notion de marque est expliquée au point 3.4.6.2.1.2. Il sera donc nécessaire dans ce cas d'effectuer le passage de la liste de paramètres de la commande à celle de l'opérateur correspondant. (*)

Développons maintenant les répercussions de cette commande sur le texte de l'opérateur en cours d'édition.

A partir de cette commande, le système génèrera dans le texte en langage LG, une instruction opérateur (en fin du texte) dont

- le <qualificateur d'opérateur> prendra la valeur de <nom dessin> contenu dans la commande TRACER si ∃

  Si ∄ on prendra le nom généré d'office (nom opérateur + numéro sans dépasser 8 caractères)

- l'<identificateur opérateur> prendra la valeur de <nom opérateur> de la commande TRACER.

- la <liste de paramètres effectifs> vaudra

  - soit la <liste de paramètres effectifs> de la commande TRACER si <nom opérateur> correspond au nom d'un opérateur composé.
    (nous ne sommes pas dans le cas (*) le passage de paramètres entre la commande et l'opérateur composé est direct).

  - soit les valeurs délivrées après analyse
    (nous sommes dans le cas (*))

- le <modificateur> sera vide.

I.2.4.2. Les commandes opérations.

- <nom-opération> <liste de paramètres effectifs> :
  <liste de nom de dessin> avec <liste de nom de dessin>
  :: = <nom dessin> { - <nom dessin>

Effectue l'opération de nom "nom - opération" de la manière définie par la "liste de paramètres effectifs" sur la réalisation d'un opérateur désigné par "liste de "nom de dessin".

Condition : - Nom - opération ∃ dans le système.
- la liste de paramètres effectifs doit correspondre à la liste des paramètres formels de l'opération désignée par nom-opération.
- que la "liste de nom de dessin" représente un chemin partant de la racine juqu'au noeud de la structure du dessin support de l'édition. Ce noeud correspond au dessin sur lequel on désire que l'opération porte.

Remarque : l'utilisateur aura la possibilité de désigner l'élément sur lequel il désire effectuer une opération suivant les manières expliquées précédement (pt 3.4.6.2.1.2.)

La liste des opérations disponibles est la suivante :

- ROTATION
- TRANSLATION
- SYMETRIE - POINT
- SYMETRIE - DROITE
- REDUCTION
- DILATATION
- COULEUR
- TRAIT

Analysons enfin les répercussions de cette commande sur le texte de l'opérateur en cours d'édition.

A partir de cette commande, le système génèrera dans le texte, en langage LG, un modificateur.

Si <liste de nom de dessin> désigne la racine de la structure du dessin support de l'édition en mode graphique

alors à partir de cette commande, le système génère dans le texte en langage LG, un modificateur global pour chaque instruction opérateur de ce texte.
Le modificateur global aura un

(*) <ident opération> égal à <nom - opération> de la cde et une <liste de paramètre effectif> égale à la <liste de paramètres effectifs> de la cde.

sinon Si <liste de nom de dessin> désigne un composant (niveau 2) de la structure du dessin support de l'édition en mode graphique

alors à partir de cette commande, le système génère dans le texte en langage LG, un modificateur global pour l'instrument opérateur dont le <qualificateur d'opérateur> correspond à la <liste de nom de dessin> . Le modificateur global sera construit de manière similaire à (*).

sinon le système détermine si dans le texte en langage LG, dans l'instruction opérateur définissant le sous-arbre de la structure de l'opérateur

en cours d'édition contenant le noeud associé au dessin constituant (niveau > 2) de la structure du dessin support sur lequel on veut agir, il existe un modificateur local le concernant.
(cette correspondance est expliquée
- au point 3.2.3. correspondant structure famille et structure d'un dessin de la famille.
- au point 3.3.1.1. correspondant structure famille et structure de l'opérateur la décrivant).

si oui

le système ajoute en fin de ce modificateur une opération dont <identificateur d'opération> sera
(xx) égal à <nom d'opération> de la commande et <liste de paramètres effectifs> identique à celle de la commande.

si non

le système génère pour cette instruction opérateur, un modificateur local.
Le modificateur local aura un <identification opération> égal à <nom d'opération> de la commande, une <liste de paramètres effectifs> égale à la <liste de paramètres effectifs> de la commande, et une <liste de qualificateur> égal à la <liste de nom de dessin> de la commande moins le premier nom de dessin de cette liste.

(Nous avons fait cette distinction afin de respecter la règle qui dit que l'on ne peut trouver dans une même instruction opérateur, 2 modificateurs locaux possédant la même liste de qualificateurs d'opérateur).

- SUPPRIMER <nom dessin>
  <nom dessin> :: = <qualificateur>

Effectue la suppression du dessin correspondant à un noeud de nom "nom dessin" de la sous-structure du dessin support de l'édition.

Condition : Nom - dessin représente le nom d'un noeud de la sous-structure du dessin support de l'édition.

analysons les répercussions de cette commande sur le texte de l'opérateur en cours d'édition.

Si <nom de dessin> désigne la racine de la structure du dessin support de l'édition en sous-mode graphique
alors le système supprime tout le corps du texte ainsi que la <liste de paramètres formels> se trouvant dans l'en-tête et par conséquent, le dessin correspondant à l'écran.
sinon
Si <nom de dessin> désigne un composant (niveau 2) de la structure du dessin support de l'édition en sous-mode graphique.
alors le système supprime dans le texte l'instruction opérateur dont le <qualificateur d'opérateur> correspond au <nom de dessin> et par conséquent le dessin correspondant à l'écran.
sinon message d'erreur.

- Remarque (concernant les opérations non transférables couleur et trait, cfr 33124 b).

Chaque fois que nous avons indiqué que le système générait un modificateur global ou un modificateur local (opération), il

est possible de vérifier si dans l'instruction opérateur, concernée, il n'existe pas déjà un ou plusieurs modificateurs globaux contenant la même opération, ou plusieurs occurences de la même opération dans ce modificateur local.
Dans l'affirmative, au lieu de générer pour un modificateur global ou pour un modificateur local une opération, on peut simplement changer la valeur de la dernière occurence d'opération, et ceci afin de ne pas surcharger inutilement le texte avec un grand nombre de modificateurs contenant la même opération

I.2.4.3. Commandes états

<NOM- ETAT> <liste de paramètres effectifs>
Définit l'état de nom "NOM - ETAT" de la manière donnée par la <liste de paramètres effectifs>

Condition : - Etat doit ∃ dans le système.
- la liste des paramètres effectifs doit correspondre à la liste des paramètres formels de l'Etat désigné par "NOM-ETAT"

La liste des Etats disponibles est la suivante :

- COULEUR
- TRAIT
- ECRAN

Analysons les répercussions de cette commande sur le texte de l'opérateur en cours d'édition.

- A partir de cette commande, le système générera dans le texte en langage LG une instruction d'état où chaque élément de l'instruction correspond à ceux de la commande. Cette instruction est placée à la fin du texte.

- Cependant, le cas de ECRAN est particulier.
Si aucune instruction d'état ECRAN n'a été définie avant la première instruction opérateur

alors le système génère dans le texte en langage LG, une instruction d'état ECRAN (dont chaque élément de l'instruction correspond à ceux de la commande), et qui sera placée dans le texte avant la première instruction opérateur.

sinon le système modifiera, dans le texte en langage LG la valeur de la dernière instruction d'état ECRAN définie avant la première instruction opérateur. La nouvelle valeur sera celle définie par la commande.

I.2.5. Définition_des_paramètres.

DEFPAR <identificateur> : = <nombre>
{ | <identificateur> : = <nombre> }

Cette commande permet à l'utilisateur de définir un ou plusieurs paramètres formels pour l'opérateur en cours d'édition ainsi que l'attribution d'une valeur à chacun de ces paramètres (utilisée dans le sous-mode graphique). Il faudra également respecter lors des utilisations successives de DEFPAR au cours d'une même édition, la contrainte que tous les paramètres formels de l'en-tête de l'opérateur doivent avoir des identificateurs différents. De plus, il faudra pour chaque paramètre ainsi défini, aller compléter la table des variables créée lors de l'entrée en sous-mode graphique.

Analysons maintenant les répercussions de cette commande sur le texte de l'opérateur en cours d'édition :

A chaque <identificateur> correspondra dans l'entête de l'opérateur en cours d'édition un paramètre formel dont <identificateur de paramètres formels> prendra comme valeur <identificateur>.

Au point de vue du texte, la valeur <nombre> n'a pas d'importance.

SUPPAR <identificateur>

Cette commande permet à l'utilisateur de supprimer un paramètre formel pour l'opérateur en cours d'édition, la table des variables étant mise à jour.

Condition : le paramètre formel à supprimer doit avoir été créé à l'aide de la commande DEFPAR lors de l'édition courante en sous-mode graphique afin d'éviter les problèmes qu'engendrerait la suppression d'un paramètre formel défini en sous-mode texte et dont la valeur aurait été modifiée par une ou plusieurs instructions d'affectation.

Ex : Maison (a------); si a prend la valeur 1, on obtient :

```
x': x(a------);          a=1
a : = a + 2;             a=3
y': y(a------);          a=3
a : = a + 1;             a=4
z': z (a-----);          a=4
b : = a + 5;
w': w (b-----).
```

D'après l'exemple précédent, la suppression en sous-mode graphique du paramètre "a" engendrerait son remplacement comme paramètre effectif variable, par sa valeur (qui est différente pour chacune des 3 instructions opérateurs dans lesquelles il apparaît comme paramètre effectif variable). Or, en sous-mode graphique, on ne dispose que de la dernière valeur de "a" à la fin de l'exécution de l'opérateur, c'est-à-dire "4" (cfr. table des variables expliquées au point 3.4.6.2.1.2.)

De plus, il faudrait supprimer les instructions d'affectation pour lesquelles "a" est membre de gauche. Il faudrait également remplacer a par sa valeur courante dans le cas où il intervient dans le membre de droite d'une instruction d'affectation (ex : b : = a + 5).

En restreignant la suppression aux paramètres formels définis par DEFPAR en sous-mode graphique, on est sûr d'éviter ce problème car un tel paramètre ne peut alors apparaître dans une instruction d'affectation à ce moment (dans l'édition en sous-mode graphique), aucun mécanisme ne permettant la création de telles instructions dans ce sous-mode.

Analysons maintenant les répercussions de cette commande sur le texte de l'opérateur en cours d'édition :

- Supprime de l'en-tête de l'opérateur en cours d'édition, le paramètre formel dont l'<identificateur de paramètres formels> à la valeur <identificateur> .
  De plus, dans le corps de l'opérateur, les paramètres effectifs variables correspondant au paramètre formel supprimé prendront la valeur qui avait été attribuée à ce paramètre formel (ils deviendront ainsi paramètres effectifs valeur).

VALPAR <identificateur>.

Cette commande permet d'obtenir la valeur de toute variable reprise dans la liste des variables créée lors de l'entrée en sous-mode graphique et mise à jour par les commandes DEFPAR et SUPPAR.
Si <identificateur> est omis, le système fournira la liste des variables de la liste des variables ainsi que pour chacune d'elle, sa valeur.

I.2.6. Marquage.

DM pt l, nom

La commande DM permet de définir une marque et éventuellement de l'identifier.
La nouvelle marque ainsi définie sera visualisée à l'écran.
pt l peut valoir ** (cfr.I.2.4.1.),
nom peut valoir rien : si on n'identifie pas la marque
* : si on désire identifier la marque. On décide qu'on ne mémorise pas plus de 10 marques identifiées de numéro 0 à 9.

Le système déterminera d'office le nom en fonction des numéros libres.
S'il ∃ 10 marques lors de la définition de la 11ème, un message préviendra l'utilisateur du fait qu'il faut faire disparaître une des 10 marques.
Il pourra choisir la marque qu'il désire faire disparaître sinon d'office la plus ancienne sera supprimée.

VM nom

La commande VM permet de visualiser une ou toutes les marques (identifiées ou non identifiées) à l'écran.
nom peut valoir * : pour toutes les marques (identifiées et non identifiées).
\+ : pour toutes les marques identifiées.
\- : pour toutes les marques non identifiées.
num : pour une marque identifiée de numéro num (0 à 9).
A : pour l'avant dernière marque non identifiée.
B : pour la dernière marque non identifiée.

Remarque : on ne mémorise que 2 marques non identifiées.

EM nom

La commande EM permet d'effacer une ou plusieurs marques (identifiées ou non identifiées) de l'écran.
nom peut prendre les même valeurs que dans la commande VM.

SM nom

La commande SM permet de supprimer une ou plusieurs marques (identifiées ou non identifiées).
nom peut prendre les mêmes valeurs que dans la commande VM.

I.2.7. Curseur

Déplacement du curseur

Il suffit de taper sur la touche
A : pour aller vers le bas
Q : pour aller vers le haut
O : pour aller vers la gauche
P : pour aller vers la droite.

Le déplacement du curseur sera fonction également du pas qui a été défini.

PD num

Cette commande permet de définir un nouveau pas de déplacement du curseur.
num : nombre.

POS pt

Cette commande permet de se positionner en un point particulier de l'écran.
pt peut prendre les valeurs ** (cfr. I.2.4.1.)

I.2.8. ECRAN.

Déplacement de l'écran dans l'espace dessin :

H : déplacement d'un demi-écran vers le haut
G : déplacement d'un demi-écran vers la gauche
D : déplacement d'un demi-écran vers la droite
B : déplacement d'un demi-écran vers le bas.

Visualisation d'une plus grande partie du dessin à l'écran.

AGRFEN cette commande permet d'obtenir une vision plus éloignée du dessin.

Visualisation d'une plus petite partie du dessin à l'écran.

DIMFEN cette commande permet d'obtenir une vision plus approchée du dessin.

I.2.9. Sauvetage.

SAVE permet de sauver le texte de l'opérateur, texte généré par le système.

I.2.10. Commande de renseignements sur la structure.

STR permet d'obtenir l'arbre de structure de l'opérateur en cours d'édition. Cette structure sera toujours disponible.

II. Les commandes du mode fichier - opérateur.

AFT <nom opérateur>

La commande AFT affiche à l'écran le texte de l'opérateur de nom NOM OPERATEUR.
Le texte de l'opérateur est constitué par le contenu de

<en-tête d'opérateur> <bloc>

Condition : NOM OPERATEUR ∃

SUPPOP <nom opérateur>

La commande SUPPOP rend l'opérateur de NOM OPERATEUR inaccessible pour une utilisation ultérieure dans le cadre de la création d'un nouvel opérateur et la modification d'un opérateur existant, tout en le laissant disponible aux utilisations antérieures.
Condition : NOM OPERATEUR ∃

IMPT <nom opérateur> ,n

La commande IMPT imprime le texte de l'opérateur de nom "NOM OPERATEUR" en un nombre d'exemplaires déterminé par "n".
Le texte de l'opérateur est constitué par le contenu de

<en-tête d'opérateur> <bloc>

<n> :: = <chiffre> { <chiffre> }

Condition : nom opérateur
n entier > 0

VS <nom-opérateur>

La commande VS permet d'effectuer une vérification de la syntaxe du texte de l'opérateur
de nom " <nom opérateur> "

Elle communique une liste des erreurs ainsi que l'origine de celles-ci s'il en ∃ .

Condition : nom opérateur ∃ dans le système.

RENAME OPER Ancien nom-opérateur TO Nouveau-nom-opérateur

La commande RENAME OPER permet de changer le nom 'ANCIEN NOM OPERATEUR" de l'opérateur par le nom "NOUVEAU NOM OPERATEUR".

Condition : . La valeur de "ANCIEN NOM OPERATEUR" ∃ dans le système.
. La valeur de "NOUVEAU NOM OPERATEUR" ∄ dans le système.

<ANCIEN NOM OPERATEUR> :: = <identificateur>
<NOUVEAU NOM OPERATEUR> :: = <identificateur>

COPIEOPER Nom-opérateur-1 TO Nom-opérateur-2

La commande COPIEOPER permet de créer une nouvelle version de l'opérateur de nom "NOM OPERATEUR - 1" version qui portera le nom "NOM OPERATEUR - 2".

Condition : La valeur de "NOM OPERATEUR - 1" ∃ dans le système.
La valeur de "NOM OPERATEUR - 2" ∄ dans le système.

<Nom-opérateur - 1> :: = <identificateur>
<Nom-opérateur - 2> :: = <identificateur>

CHARGER <liste de nom opérateur>

La commande "CHARGER" permet à l'utilisateur de charger les textes des opérateurs de nom <liste de nom opérateur> en mémoire centralé.

<liste de nom opérateur> :: = <nom opérateur> { , <nom opérateur> }

Condition : <nom opérateur> désigne un opérateur existant dans le système, et dont le texte est non encore chargé en mémoire.

III. Les commandes du mode fichier - dessin

CREERDESSIN <nom opérateur> <liste paramètres effectifs>

<modificateur> : <nom dessin>

La commande CREER-DESSIN permet de créer un dessin de nom "nom dessin" à partir d'un opérateur "nom-opérateur" dont les caractéristiques sont définies par les paramètres "paramètres" liés à cet opérateur et les Modificateurs "modificateur ".

<nom Dessin> :: = <identificateur>

Un affichage de la représentation graphique du dessin est également effectué.

Condition : - nom opérateur ∃ dans le système.
- le nombre d'élément dans "liste paramètres effectifs" doit correspondre au nombre de paramètres formels liés à l'opérateur de nom "nom-opérateur".
- nom-Dessin ∄ dans le système.

AFFDESSIN <nom Dessin>

La commande AFFDESSIN permet d'afficher un dessin de nom "nom dessin" : <nom dessin> :: =<identificateur> .
Il se peut qu'une partie du dessin soit affichée , de ce fait, l'utilisateur dispose d'un ensemble de commande de type ECRAN lui permettant de se déplacer dans son dessin. Une description des commandes de type "ECRAN" sera effectuée par la suite.
Condition : nom dessin ∃ dans le système.

SUPP DESSIN <nom Dessin>

La commande SUPP DESSIN permet de supprimer un dessin de nom "nom dessin".

<nom dessin> :: = <identificateur>

Condition : nom dessin ∃ dans le système.

RENAMEDESSIN <ancien NOM-DESSIN> TO <nouveau NOM-DESSIN>

La commande RENAMEDESSIN permet de changer le nom d'un dessin de nom "ancien nom-dessin" par le nom "nouveau nom-dessin".

Condition : ancien-nom-dessin ∃ dans le système.
nouveau-nom-dessin ∄ dans le système.

IMPDESSIN <nom Dessin>

La commande IMPDESSIN permet d'imprimer un Dessin de nom "nom dessin"
Si "nom dessin" est vide, alors il désigne le dessin courant.

<nom dessin> :: = <identificateur> | <vide>

Dans le cas où "nom dessin" est donné, le dessin sera entièrement imprimé ce qui signifie que la surface de l'écran n'est pas la référence pour l'espace imprimante c'est-à-dire que si seulement une partie d'un dessin est visible à l'écran, sur imprimante ce sera le dessin qui sera imprimé, d'où nécessité de parfois réduire le dessin afin de l'ajuster à l'espace imprimante.

Dans le cas où "nom dessin" est vide, seule la partie du dessin affichée à l'écran sera imprimée. Ici, il y aura donc correspondance entre l'affichage et l'impression.

Condition : "nom dessin" ∃ dans le système.

Si "nom dessin" = <vide> il faut qu'il ∃ un dessin courant.

Les commandes de type ECRAN

Les commandes sont identiques à celles définies dans le sous-mode graphique (cfr. annexe G. point I.2.8.).

VI. Les commandes du mode Aide.

(AI) AFINF <sujet>

Affiche des informations concernant le sujet déterminé par "sujet".

<sujet> :: = OPERATEUR | OPERATION | ETAT | COMMANDE | DESSIN

OPERATEUR signifie que l'on désire obtenir des informations sur les opérateurs existans dans le système.
Les informations auront la forme :

nom opérateur, liste des paramètres formels.

OPERATION
ETAT
COMMANDE
DESSIN

} même forme et signification que les opérateurs.

<nom opérateur> | <nom opération> | <nom état> | <nom commande>

Dans les 4 cas, on désigne un élément de l'ensemble des opérateurs, opérations, état, commandes.

Informations fournies : -nom operateur, liste de paramètres formels
opération
état
commande

Condition : Sujet ┤

(II) IMPINF <sujet>

Imprime des informations concernant le sujet déterminé par "sujet".

<sujet> :: = OPERATEUR | OPERATION .... idem que dans AFINF

Condition : Sujet ┤

V. La commande du mode Initialisation

(EC) ECHELLE Rapport

Cette commande permet d'exprimer le rapport entre l'axe des Y et l'axe des X, c'est-à-dire le rapport du pas en Y en fonction du pas en X. Ce rapport est exprimé par une valeur positive entre 0 et 1 donné par "Rapport". Au départ, le rapport est fixé (valeur par défaut).

Cette commande est utile lorsque l'on désire trouver le même dessin sur différents terminaux.

Ex : Lorsque vous tracez un cercle, celui-ci peut être plus ou moins applati ou rétréci suivant le terminal. Grâce à cette commande, on pourra adapter le dessin afin d'obtenir la même représentation du cercle au terminal.

ANNEXE H : Le ruissellement des modificateurs

- Algorithme
- Exemple

I. ALGORITHME

I.1. Explication du fonctionnement de l'algorithme du ruissellement des modificateurs.

Rappelons que :

- Cet algorithme de ruissellement est exécuté lors de la phase "exécution" d'un opérateur, c'est-à-dire suite à la commande TRACER dans le mode graphique, suite au passage du mode texte au mode graphique grâce à la commande GOGRAF et lors de l'entrée en mode graphique par l'instruction EDG (Editgraph).
  Ajoutons que le ruissellement se fera également à la suite de la commande du mode fichier-dessin : "CREERDESSIN".
  Mais, comme ce mode ne fait pas l'objet d'une étude approfondie, nous ne ferons ici que la citer.

- Cet algorithme voit le texte comme une suite d'instructions. En fait, celui-ci travaille instruction par instruction.

- Le but de l'algorithme est de fournir pour chaque opérateur de base; le produit de composition des opérations transférables, la bonne valeur des opérations non-transférables liées à ceux-ci, ainsi que la valeur de l'état écran, globale à tous ces opérateurs.
  Le produit de composition et les bonnes valeurs étant déterminés grâce aux règles de priorités définies dans l'héritage des modificateurs au point 3.3.1.2.4. de l'analyse fonctionnelle.

Spécification de l'algorithme.

En entrée

- On dispose d'une commande d'un des 4 types suivants :
  - TRACER
  - GOGRAPH
  - EDG
  - CREERDESSIN

  permettant d'identifier l'opérateur pour lequel on va effectuer le ruissellement des modificateurs.
  A ce sujet, nous ferons les hypothèses simplificatrices suivantes :
  - Cette commande aura une syntaxe correcte et sa forme sera celle spécifiée dans l'annexe G.
  - On dispose en M.C. des textes des opérateurs nécessaires, chaque texte ayant une syntaxe correcte, donc, conforme à la syntaxe BNF du langage LG définie en annexe B.

Objectif

Fournir pour chaque opérateur de base associé à chaque feuille de la structure de l'opérateur désigné par la commande, le produit de composition des opérations transférables ainsi que la bonne valeur pour la couleur et le trait. De plus, il fournit la bonne valeur pour le fond de l'écran, celle-ci étant globale à toutes opérations de base.

Sortie

- La bonne valeur pour le fond de l'écran.
- Pour chaque opérateur de base :
  a) Une liste d'enregistrements dans laquelle chacun d'eux aura la forme suivante :

NN	Valeur de l'opération	type	NN provenance

NN étant le n° de niveau du noeud à partir duquel porte la "valeur de l'opération " de type "type", la valeur de l'opération ayant été définie dans une instruction opérateur dont le noeud lui étant lié dans la structure de l'opérateur est de niveau "NN provenance".

Chaque enregistrement contiendra une opération transférable et l'ordre de rangement sera déterminé grâce aux règles de priorité définies dans l'héritage des modificateurs au point 3.3.1.2.4. de l'analyse fonctionnelle.

b) La bonne valeur pour le trait et la couleur.

Remarque :

Par manque de temps, nous n'avons rien analysé au niveau du stockage des informations manipulées par l'éditeur graphique. Suite à cela, nous ne pourrons donner aucun renseignement valable concernant l'organisation physique en mémoire de la commande et des textes des opérateurs manipulés par cet algorithme.

Définition des tables utilisées par l'algorithme.

Pour réaliser son objectif, l'algorithme fonctionnera de la façon suivante :

- Il puisera l'ensemble des informations nécessaires à son fonctionnement :
  - dans les modificateurs locaux et globaux qu'il trouvera dans chaque instruction opérateur;
  - dans les instructions d'état de type couleur, trait et écran.

- Il retiendra toutes ces informations dans des tables qui au fur et à mesure du traitement des instructions s'échangeront celles-ci de telle manière que l'objectif concernant les opérateurs de base soit rempli.

  Ces tables sont au nombre de 6 et sont libellées de la manière suivante :

  - la table des couleurs
  - la table des traits
  - la table des transférables
  - la table des locaux couleurs
  - la table des locaux traits
  - la table des locaux transférables.

Remarque :

Nous considérons à partir de ce moment qu'une table est constituée d'un ensemble d'enregistrements.

a) La table des couleurs. (table couleur)

Cette table contiendra après chaque instruction opérateur traitée, la bonne valeur "couleur" pour le noeud lié à cette instruction. Cette valeur se trouvera dans le dernier enregistrement de la table.

Si tel est le cas pour chaque noeud de la structure, alors il le sera pour les feuilles qui sont liées aux opérateurs de base. Donc pour chaque opérateur de base, on déterminera correctement la valeur de la "couleur associée". Les enregistements de cette table auront la forme suivante:

NN	Valeur de la couleur	type

- NN étant le n° de niveau du noeud sur lequel porte la "valeur de la couleur".
- "Type" détermine si cette valeur provient d'une instruction état ou d'un modificateur.

b) La table des traits.(table trait)

Cette table contiendra après chaque instruction opérateur traitée, la bonne valeur "trait" pour le noeud lié à cette instruction. Cette valeur se trouvera dans le dernier enregistrement de la table. Si tel est le cas pour chaque noeud de la structure, alors il le sera pour les feuilles qui sont liées aux opérateurs de base. Donc pour chaque opérateurs de base, on déterminera correctement la valeur de "trait" associée.

Les enregistrements de cette table auront la forme suivante :

NN	Valeur du trait	Type

- NN étant le n° de niveau du noeud sur lequel porte la "valeur du trait".
- "Type" détermine si cette valeur provient d'une instruction état ou d'un modificateur.

c) La table des transférables. (table transférables)

Cette table contiendra après chaque instruction opérateur traitée, la liste des opérations transférables pour le noeud lié à cette instruction. Cette liste permettra de construire le produit de composition des opérations transférables. De plus, cette construction sera facilitée par le rangement dans le bon ordre des opérations dans la table (chaque enregistrement correspondant à une opération;

la lecture s'effectuant du début de la table jusqu'à la fin de celle-ci).
Si tel est le cas pour chaque noeud de la structure, alors il le sera pour les feuilles qui sont liées aux opérateurs de base. Donc, pour chaque opérateur de base, on déterminera correctement le produit de composition associé.

Les enregistrements de cette table auront la forme suivante :

NN	Valeur de l'opération	Type	NN provenance

- NN étant le n° de niveau du noeud à partir duquel porte la"valeur de l'opération" de type "type", la valeur de l'opération ayant été définie dans une instruction opérateur dont le noeud lui étant lié dans la structure de l'opérateur est de niveau "NN provenance".

  En fait, NN spécifie le n° de niveau du noeud sur lequel porte l'opération. De plus, cette opération porte sur le sous-arbre défini dans la structure de l'opérateur et dont la racine est ce noeud. Ceci, afin de préciser dans la définition de NN donnée ci-dessus, la signification du groupe de mot "à parir duquel".

  Ajoutons que "type" spécifie une opération telle que : translation, rotation, symétrie, réduction et dilatation et que "NN provenance" permettra, lors de l'édition en mode graphique , dans le cas d'une représentation intermédiaire, d'insérer au bon endroit dans le produit de composition , l'information correspondant à l'instruction opération portant sur un noeud particulier de la structure. De plus amples explications à ce niveau seront fournies au point 4.5.2.3.3.

d) La table des locaux couleurs. (table local couleur)

Cette table contiendra après chaque instruction opérateur traitée, la liste des locaux de type couleur qui porteront sur des noeuds appartenant au sous-arbre défini dans la Structure de l'opérateur et ayant pour racine, le noeud correspondant à l'instruction opérateur d'où ils proviennent. Ces informations seront prises en compte, c'est-à-dire transférées dans la table "couleur", lorsque l'instruction, liée au noeud sur lequel elles portent, sera traitée.

Les enregistrements de cette table auront la forme suivante :

NN	Valeur de la couleur	liste de qualificateurs	NN provenance

- NN étant le n° de niveau du noeud sur lequel porte la "valeur de la couleur". Ce noeud est désigné par la "liste de qualificateurs" et "NN provenance" spécifie le n° de niveau du noeud lié à l'instruction opérateur où a été définie l'occurence de l'opération couleur.

  Précisons que "NN provenance" permettra de résoudre le problème suivant :

  - Lorsque l'on rencontre dans un même modificateur local plusieurs opérations du type couleur, seule la dernière est prise en compte.

  Ajoutons que cette table est en ordre croissant sur le "NN".

e) La table des locaux Traits. (table local trait)

Cette table contiendra après chaque instruction opérateur traitée, la liste des locaux de type trait qui porteront sur des noeuds appartenant au sous-arbre défini dans la structure de l'opérateur et ayant pour racine le noeud correspondant à l'instruction opérateur d'où ils proviennent. Ces informations seront prises en compte, c'est-à-dire transférées dans la Table Trait, lorsque l'instruction liée au noeud sur lequel elles portent, sera traitée.

Les enregistrements de cette table auront la forme suivante :

NN	Valeur de trait	liste de qualificateur	NN provenance

- NN étant le n° de niveau du noeud sur lequel porte la"valeur de trait". Ce noeud est désigné par la "liste de qualificateurs" et "NN provenance" spécifie le n° de niveau du noeud lié à l'instruction opérateur où a été définie l'occurence de l'opération trait.

  Précisons que "NN provenance" permettra de résoudre le problème suivant :

  - Lorsque l'on rencontre dans un même modificateur local plusieurs opérations du type trait, seule la dernière est prise en compte.

  Ajoutons que cette table est en ordre croissant sur le "NN".

f) La table des locaux transférables.(table locaux transférables)

Cette table contiendra après chaque instruction opérateur traitée, la liste des locaux de type transférable qui porteront sur des noeuds appartenant au sous-arbre défini dans la structure de l'opérateur et ayant pour racine, le noeud correspondant à l'instruction opérateur d'où ils proviennent. Ces informations seront prises en compte, c'est-à-dire transférées dans la table des transférables, lorsque l'instruction liée au noeud sur lequel elles portent sera traitée.

Les enregistrements de cette table auront la forme suivante:

NN	Valeur de l'opération	type d'opération	liste de qualificateur	NN provenance

- NN étant le n° de niveau du noeud sur lequel porte la "valeur de l'opération" de type "type"d'opération. Ce noeud est désigné par la "liste de qualificateur" et "NN provenance" spécifie le n° de niveau du noeud lié à l'instruction opérateur où a été définie l'occurence de l'opération de type "type d'opération".

  Précisons que "NN provenance" n'a aucun intérêt à ce niveau, mais comme celui-ci sera nécessaire dans la table des transférables et qu'entre celle-ci et la table des locaux transférables des échanges se produiront, on retiendra le "NN provenance" dans les enregistrements de cette dernière.

  Ajoutons que cette table est en ordre croissant sur le "NN".

Rappelons enfin que tout échange entre les tables, et

tout ajout d'information concernant les occurences d'opération dans celles-ci seront effectués en respectant les règles de priorité définie au point 3.3.1.2.4. dans l'analyse fonctionnelle.

Signalons également que toutes commandes du type TRACER, GOGRAF et CREERDESSIN seront traitées comme des instructions opérateur.

Enfin, précisons que la bonne valeur du fond de l'écran se trouve dans la variable ECRAN.

Avant de présenter l'algorithme, définissons les variables que celui-ci utilise :

NNC :
Le numéro de niveau courant dans la structure de l'opérateur.

NIO :
Indique si nous avons rencontré ou non une instruction opérateur (=1 dans le cas favorable, =0 si non).

NN couleur :
Donne le nombre d'enregistrements dans la table couleur.

NN trait :
Donne le nombre d'enregistrements dans la table trait.

NN local Trait, NN local Couleur, NN local Transférable :
Tous 3 donnent le nombre d'enregistrements dans leur table correspondante, c'est-à-dire la table local trait, la table local couleur et la table local transférable.

Nom - Qualificateur - Courant :
Donne le nom du noeud dans la structure de l'opérateur correspondant à l'instruction opérateur que l'on traite.
Il est représenté sous la forme d'une liste de qualificateurs.

NN provenance dernier.
Variable intermédiaire contenant la valeur courante du NN provenance du dernier record de Table local couleur.

NN du dernier record (enregistrement) de table couleur :
Variable intermédiaire contenant la valeur courante du NN du dernier record de la Table couleur.

Liste qualificateur du dernier record :
Variable intermédiaire contenant la valeur courante de la liste de qualificateur du dernier record de la table couleur.

NN du dernier record (enregistrement) de table trait :
Variable intermédiaire contenant la valeur courante du NN du dernier record de la Table trait.

NN premier enregistrement de Table local couleur :
Variable intermédiaire contenant la valeur courante du NN du premier enregistrement de la Table local couleur.

NN premier enregistrement de Table local trait :
Variable intermédiaire contenant la valeur courante du NN du premier enregistrement de la Table local trait.

Note concernant la lecture d'une commande et d'une instruction.

La lecture consiste en un scanning d'une commande ou d'une instruction, à partir duquel on créera deux tables : la table des modificateurs locaux, la table des modificateurs globaux.
La première contiendra tous les modificateurs locaux définis dans la commande ou l'instruction que l'on lit, et dans l'ordre d'apparition de ceux-ci.
La deuxième contiendra tous les modificateurs globaux définis dans la commande ou l'instruction que l'on lit, et dans l'ordre d'apparition de ceux-ci.

Ces tables seront constituées d'enregistrements ayant la forme suivante :

a) Pour la table des modificateurs locaux.

type d'opération	valeurs de l'opérateur	liste de qualificateur

b) Pour la table des modificateurs globaux.

type d'opération	valeur de l'opérateur

Ajoutons que lorsque l'on rencontrera :
"tant qu'il ∃ MG"
ou
"tant qu'il ∃ ML" dans l'algorithme,
Cela signifiera : tant qu'il ∃ des enregistrements dans la table des modificateurs globaux.

tant qu'il ∃ des enregistrements dans la table des modificateurs locaux.

A la fin du traitement de chaque instruction, ces tables sont détruites.

I.2. Algorithme de ruissellement des modificateurs.

Traitement général

- Initialisation
- Traitement de la commande

Initialisation

NNC = 1
NIO = Ø
Ecran = Valeur par défaut écran
Couleur = Valeur par défaut couleur
Trait = Valeur par défaut trait
NN couleur = Ø
NN trait = Ø
NN local trait = Ø
NN local couleur = Ø
NN local transférable = Ø
NN transférable = Ø
Nom - Qualificateur - courant = Ø

Traitement de la commande

- Lire la commande.
- Traitement Instruction Opérateur.
- Se positionner au début du bloc du texte de l'opérateur désigné par la commande.
- Traitement opérateur.

Traitement opérateur

NNC = NNC + 1
tant qu'il ∃ des instructions dans le bloc
    lire l'instruction suivante
    traitement instruction
NNC = NNC - 1

Traitement instruction

Si NN couleur ≠ Ø alors

Si NNC ⩽ NN du dernier record de table couleur

alors Supprimer records de table couleur dont NN ⩾ NNC sauf si = état et
A chaque suppression faire :
NN couleur = NN couleur - 1

Si NN trait ≠ Ø alors

Si NNC ⩽ NN du dernier record de table trait

alors Supprimer records de table trait dont NN ⩾ NNC sauf si = état et
A chaque suppression faire :
NN trait = NN trait - 1

Si NN transférable ≠ Ø alors

Si NNC ⩽ NN du dernier enregistrement de table transférables

alors Supprimer records de table transférables dont NN ⩾ NNC
et A chaque suppression faire :
NN transférable = NN transférable -1

Si Instruction = "ETAT" alors Traitement ETAT

Si Instruction = "OPERATEUR" alors

- ajouter à Nom-Qualificateur-courant le qualificateur de l'instruction opérateur.
- Traitement Instruction opérateur.
- NIO = 1
- Si pas opérateur de base
  alors se positionner au début du bloc de texte de l'opérateur désigné par l'instruction.
  Traitement opérateur.

  sinon Sauver pour l'opérateur de base :
  - la valeur se trouvant dans le dernier enregistrement de table trait et de table couleur.
  - les enregistrements dans table transférables en respectant l'ordre. Ceux-ci représentant le produit de composition.
- retirer à Nom-Qualificateur-courant le dernier qualificateur.

Traitement Etat

```
Si Etat = "écran" alors
                  Si NNC = 2 et NIO = Ø
                     alors Ecran = valeur de état dans
                                   l'instruction.
Si Etat = "couleur" alors
                  Si NN Couleur = Ø
                     alors - Ajouter record état dans table
                             couleur avec NN = NNC.
                           - NN Couleur = NN Couleur + 1
                     sinon
                      Si NN du dernier record dans table cou-
                         leur = NN et état
                         alors - remplacer la valeur du
                                 record de type Etat et de
                                 NN = NNC dans table couleur
                                 par la valeur dans l'instruc-
                                 tion état
                         sinon - Oublier Etat Couleur.
```

Si Etat = "trait" alors si NN trait = ∅

alors - Ajouter record état dans Table trait avec NN = NNC

- NN trait = NN trait + 1

sinon

Si NN du dernier record dans Table couleur = NNC et état

alors- remplacer la valeur du record de type état et de NN = NNC dant table couleur par la valeur dans l'instruction état

sinon- Oublier état Trait.

Traitement instruction opérateur

Traitement Ajout des Locaux Transférables aux globaux Transférables.

Tant qu'il ∃ MG alors Traitement Global

Tant qu'il ∃ ML alors Traitement Local.

Traitement Ajout des Locaux Non-Transférables aux globaux Non-Transférables.

Traitement ajout des Locaux Non-Transférables aux globaux Non-Transférables

Si NN local Couleur ≠ ∅

alors Si ∃ un enregistrement dans table Local couleur dont NN = NNC et que liste qualificateur entière dans cet enregistrement correspond en partie ou entièrement, en commençant par la fin, au Nom - Qualificateur - courant

alors

- Ajouter dans Table couleur le record de Table Local Couleur, dont on a retiré NN provenance.

- NN Couleur = NN Couleur + 1
- Supprimer de Table Local couleur le record.
- NN local Couleur = NN local couleur - 1

Si NN local Trait ≠ ∅

alors Si ∃ un enregistrement dans table Local trait dont NN = NNC et que liste qualificateur entière dans cet enregistrement correspond en partie ou entièrement, en commençant par la fin, au Nom - Qualificateur - courant

alors

- Ajouter dans Table Trait le record, de Table Local trait dont on a retiré NN provenance.
- NN trait = NN trait + 1
- Supprimer de Table Local trait le record.
- NN local trait = NN local trait - 1

Traitement ajout des Locaux Transférables aux globaux Transférables.

Si NN Local Transférable ≠ ∅

alors tant qu'il ∃ dans table Locaux Transférables un enregistrement où NN = NNC et que liste qualificateur entière dans cet enregistrement correspond en partie ou entièrement, en commençant par la fin, au Nom - Qualificateur - courant

alors

- Ajouter en fin de table transférable le record de Table locaux transférables.

- NN Transférable = NN Transférable + 1
- Supprimer de Table Locaux Transférables le record.
- NN local Transférable = NN local Transférable - 1

Traitement Local

Si ML du type Transférable alors - Ajouter dans Tables Locaux transférables le modificateur local avec NN = NNC + nombre de qualificateurs et NN provenance = NNC.
- NN local transférable = NN local transférable + 1
- Cet ajout s'effectue par ordre croissant sur le NN et NN provenance.

Si ML du type couleur alors

Si NN couleur = Ø alors NN du dernier record de Table couleur = Ø

sinon NN du dernier record de Table couleur = Valeur du NN du dernier record de table couleur.

Si NN local couleur = Ø

alors NN provenance dernier = Ø
liste qualificateur du dernier record = Ø

sinon-NN provenance dernier = Valeur du NN provenance du dernier record de Table local couleur.
-liste qualificateur du dernier record = Valeur de la liste qualificateur du dernier record de table couleur.

Si (NNC > NN du dernier record de Table couleur et NN couleur ≠ ∅)

alors pas retenir ML

sinon Si ∃ un record pour lequel, dans table locale couleur, NN provenance ≠ NNC et que NNC + nombre de qualificateur du local = NN du record et que liste qualificateur entière de ce record correspond entièrement ou en partie en commençant par la fin, au Nom - Qualificateur - courant + liste qualificateur du local dans instruction.

alors pas retenir ML

sinon Si NN provenance = NNC et que liste qualificateur du dernier record de table locale trait = liste qualificateur du local dans Instruction

alors- remplacer la valeur du dernier record de la table locale couleur par la valeur du ML.

sinon - Ajouter ML dans table local couleur avec NN = NNC + nombre de qualificateur.

- NN provenance = NNC.
- NN local couleur = NN local couleur + 1.
- Cet ajout s'effectue par ordre croissant sur le NN et NN provenance.

Si ML du type trait alors

Si NN trait = ∅ alors NN du dernier record de table trait = 0

sinon NN du dernier record de table trait = Valeur du NN du dernier record de table trait.

Si NN local trait = ∅

alors - NN provenance dernier = ∅
- Liste qualificateur du dernier record = ∅

sinon - NN provenance dernier = Valeur du NN provenance du dernier record de table local trait.
- Liste qualificateur du dernier record = Valeur de la liste qualificateur du dernier record de table local trait.

Si (NNC > NN du dernier record de table trait et NN trait ≠ ∅)

alors pas retenir ML

sinon Si ∃ un record pour lequel dans table local trait, NN provenance ≠ NNC et que NNC + nombre de qualificateur du local = NN du record et que liste qualificateur entière de ce record correspond entièrement ou en partie, en commençant par la fin, à
Nom - Qualificateur - courant + liste qualificateur du local dans instruction

alors pas retenir ML

sinon Si NN provenance dernier = NNC
et que liste qualificateur du
dernier record de table local trait
= liste qualificateur du local dans
instruction

alors - Modifier la valeur du
dernier record de table
local trait.

sinon - Ajouter ML dans table local
trait avec NN = NNC + nombre
de qualificateur
- NN provenance = NNC.
- NN local trait = NN local
trait + 1.
- Cet ajout s'effectue par
ordre croissant sur le NN
et NN provenance.

Traitement global

Si MG de type transférable alors - Ajouter dans table
transférables le MG avec
NN = NNC et NN provenance
= NNC.

Si MG de type couleur

alors

Si NN table local couleur = ∅

alors NN premier enregistrement de
table local couleur = ∅

sinon NN premier enregistrement
de table local couleur =
Valeur NN premier enregistre-
ment de table local couleur.

Si NN table couleur = ∅

alors NN dernier enregistrement table couleur = ∅

sinon NN dernier enregistrement table couleur = Valeur du NN du dernier enregistrement de table couleur.

Si ∃ dans table Local Couleur un enregistrement pour lequel NN = NNC et que la liste qualificateur entière de cet enregistrement correspond entièrement ou en partie, en commençant par la fin, à Nom - Qualificateur - courant

alors oublier MG

sinon

Si (NNC > NN du premier enregistrement de la table local couleur et NN table couleur ≠ ∅ et [(dernier enregistrement ≠ état) ou (dernier enregistrement = état et NNC ≠ NN de celui-ci)])

alors pas retenir MG

sinon si (NNC = NN du dernier enregistrement table couleur et ≠ état)

alors - remplacer la valeur du dernier record de Table couleur par la valeur dans le MG.

sinon - Inclure MG dans table couleur.
- NN table couleur = NN table couleur + 1 avec NN = NNC.

Si MG de type trait

alors

Si NN table local trait = ∅

alors NN premier enregistrement de table local trait = ∅.

sinon NN premier enregistrement de table

local trait = valeur du NN du premier enregistrement de table local trait.

Si NN table trait = ∅

alors NN dernier enregistrement table trait = ∅

sinon NN dernier enregistrement de table trait = Valeur du NN du dernier enregistrement de table trait.

Si ∃ dans table local trait un enregistrement pour lequel NN = NNC et que liste qualificateur entière de cet enregistrement correspond entièrement ou en partie, en commençant par la fin, à Nom - Qualificateur - courant

alors oublier MG

sinon Si (NNC > NN du premier enregistrement de table local trait et NN table trait ≠ ∅ et [(dernier enregistrement ≠ état) ou (dernier enregistrement = état et NNC ≠ NN de celui-ci)])

alors pas retenir MG

sinon Si (NNC = NN du dernier enregistrement table trait et ≠ état)

alors - Remplacer la valeur du dernier record de Table Trait par la valeur dans le MG.

sinon - Inclure MG dans Table Trait
- NN table trait = NN table trait + 1.

II. EXEMPLE

II.1 Commande

TRACER village (1)

le numero mis entre parentheses , est le numero de l'etape dans l'algorithme.

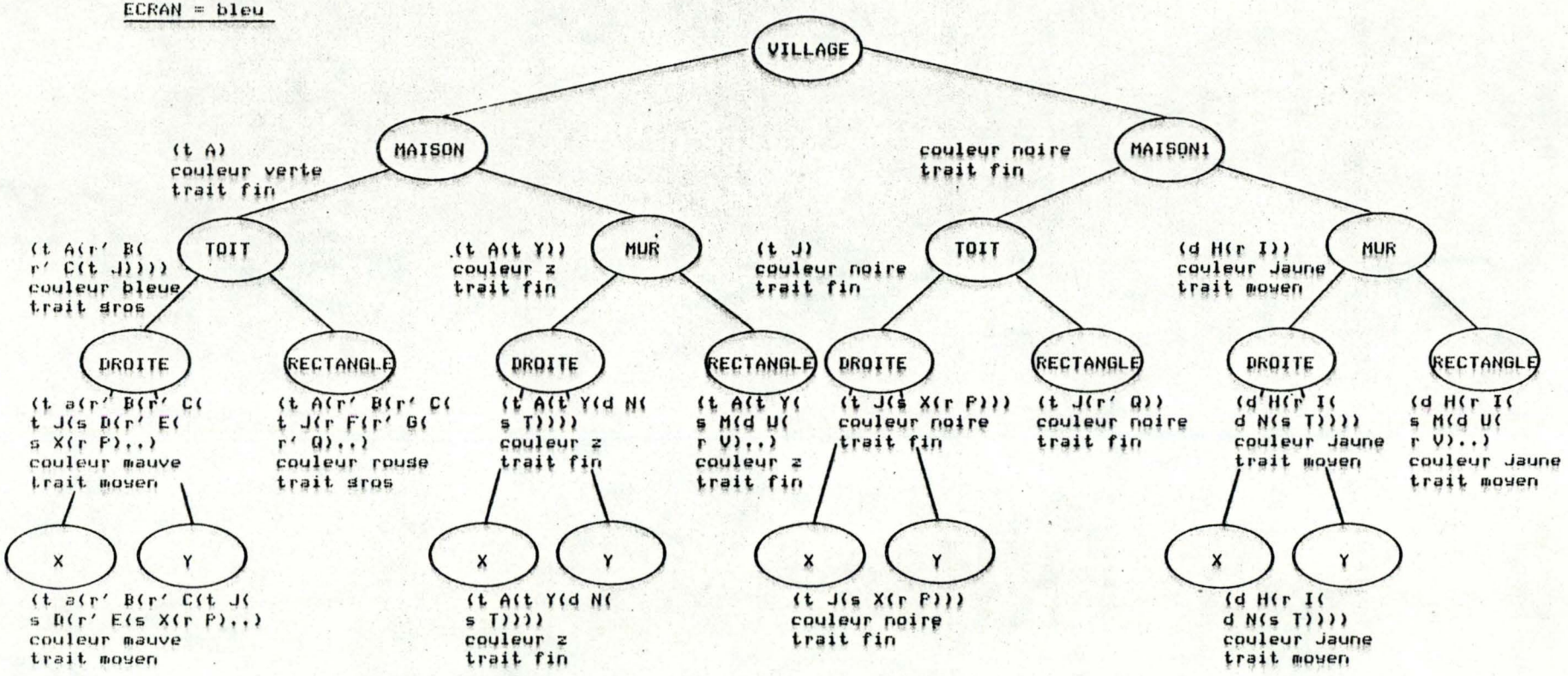
II.2 Description en langage LG des operateurs utilises

```
VILLAGE ;
        (2) ecran blanc ;
        (3) ecran bleu ;
        (4) couleur rouge ;
        (5) trait fin ;
        (6) maison : maison (a)  [couleur rouge , couleur verte,
                                    translation A ]
                                 [mur : couleur Z ,
                                        translation Y ,
                                  rectangle-toit : couleur bleu ,
                                                   couleur vert ,
                                                   reduction F ,
                                                   rotation G ,
                                                   couleur rouge ,
                                  toit : couleur bleu ,
                                         trait gros ,
                                         rotation B ,
                                         rotation C ,
                                  droite-toit : couleur mauve ,
                                                trait moyen ,
                                                symetrie D ,
                                                rotation E ] ;
       (21) ecran rouge ;
       (22) couleur noire ;
       (23) maison1 : maison (b) [mur : couleur jaune ,
                                        trait moyen ,
                                        dilatation H ,
                                        reduction I ] .


MAISON (X) ;

       (24)(7) toit : toit (c)  [ couleur blanc , trait gros ,
                                    translation J ]
                                [droite : couleur W ,
                                          symetrie X ] ;

      (32)(15) mur : mur (x) [ rectangle : couleur K ,
                                           symetrie M ,
                               droite : couleur L ,
                                        dilatation N ] .
```

```
TOIT (W) ;

        (25)(8) ecran mauve ;
        (26)(9) droite : droite (w) [ couleur rouge ,
                                      reduction P ] ;
       (30)(13) couleur cyan ;
       (31)(14) rectangle : rectangle (a) [ couleur O ,
                                            rotation Q ] .


DROITE (N) ;

        (34)(27)(17)(10) couleur blanc ;
        (35)(28)(18)(11) X : X (n) ;
        (36)(29)(19)(12) Y : Y (n) .


MUR (M) ;

        (33)(16) droite : droite (b) [ couleur R ,
                                       symetrie T ] ;
        (37)(20) rectangle : rectangle (c) [ couleur S ,
                                             dilatation U ,
                                             reduction U ,
                                             trait % ] .


OPERATEURS DE BASE

        X
        Y
        rectangle
```

RESULTATS ATTENDUS POUR CHAQUE NOEUD DE LA STRUCTURE.

ECRAN = bleu



legende : t = translation
d = dilatation
r = reduction
r' = rotation
s = symetrie

EXEMPLE ; tables 1ere partie

Etape	NNC	Table couleur	Table trait	Table transferable
1	1	VPD	VPD	
2	2	VPD	VPD	
3	2	VPD	VPD	
4	2	2 rouge ETAT	VPD	
5	2	2 rouge ETAT	2 fin ETAT	
6	2	2 rouge ETAT 2 verte	2 fin ETAT	2 A translation 2
7	3	2 rouge ETAT 2 verte 3 bleu	2 fin ETAT 3 gros	2 A translation 2 3 B rotation 2 3 C rotation 2 3 J translation 3
8	4	IDEM	IDEM	IDEM
9	4	2 rouge ETAT 2 verte 3 bleu 4 mauve	2 fin ETAT 3 gros 4 moyen	2 A translation 2 3 B rotation 2 3 C rotation 2 3 J translation 3 4 D symetrie 2 4 E rotation 2 4 X symetrie 3 4 F reduction 4
10	5	IDEM	IDEM	IDEM
11	5	IDEM	IDEM	IDEM

EXEMPLE ; tables 1ere partie

Etape	NNC	Table couleur	Table trait	Table transferable
12	5	IDEM	IDEM	IDEM
13	4	2 rouge ETAT 2 verte 3 bleu	2 fin ETAT 3 gros	2 A translation 2 3 B rotation 2 3 C rotation 2 3 J translation 3
14	4	2 rouge ETAT 2 verte 3 bleu 4 rouge	2 fin ETAT 3 gros	2 A translation 2 3 B rotation 2 3 C rotation 2 3 J translation 3 4 F reduction 2 4 G rotation 2 4 Q rotation 4
15	3	2 rouge ETAT 2 verte 3 Z	2 fin ETAT	2 A translation 2 3 Y translation 2
16	4	2 rouge ETAT 2 verte 3 Z	2 fin ETAT	2 A translation 2 3 Y translation 2 4 N dilatation 3 4 T symetrie 4
17	5	IDEM	IDEM	IDEM
18	5	IDEM	IDEM	IDEM
19	5	IDEM	IDEM	IDEM

EXEMPLE : tables 1ere partie

Etape	NNC	Table couleur	Table trait	Table transferable
20	4	2 rouge ETAT 2 vert 3 Z	2 fin ETAT	2 A translation 2 3 Y translation 2 4 M symetrie 3 4 U dilatation 4 4 V reduction 4
21	2	2 rouge ETAT	2 fin ETAT	
22	2	2 noire ETAT	2 fin ETAT	
23	2	2 noire ETAT	2 fin ETAT	
24	3	2 noire ETAT	2 fin ETAT	3 J translation 3
25	4	IDEM	IDEM	IDEM
26	4	2 noire ETAT	2 fin ETAT	3 J translation 3 4 X symetrie 3 4 P reduction 4
27	5	IDEM	IDEM	IDEM
28	5	IDEM	IDEM	IDEM
29	5	IDEM	IDEM	IDEM
30	4	2 noire ETAT	2 fin ETAT	3 J translation 3
31	4	2 noire ETAT	2 fin ETAT	3 J translation 3 4 Q rotation 4
32	3	2 noire ETAT 3 Jaune	2 fin ETAT 3 moyen	3 H dilatation 2 3 I reduction 2

EXEMPLE : tables 1ere partie

Etape	NNC	Table couleur	Table trait	Table transferable
33	4	2 noire ETAT 3 Jaune	2 fin ETAT 3 moyen	3 H dilatation 2 3 I reduction 2 4 N dilatation 3 4 T symetrie 4
34	5	IDEM	IDEM	IDEM
35	5	IDEM	IDEM	IDEM
36	5	IDEM	IDEM	IDEM
37	4	2 noire ETAT 3 Jaune	2 fin ETAT 3 moyen	3 H dilatation 2 3 I reduction 2 4 M symetrie 3 4 U dilatation 4 4 V reduction 4
38	1			

EXEMPLE : tables 2eme partie

Etape	Table locaux couleur	Table locaux trait	Table locaux transferables	Ecran
1				VPD
2				Blanc
3				bleu
4				bleu
5				bleu
6	3 Z Mur 2 3 Bleu Toit 2 4 Rouge Rectangle - toit 2 4 Mauve Droite - toit 2	3 Gros Toit 2 4 Moyen Droite -toit 2	3 Y Translation Mur 2 3 B Rotation Toit 2 3 C Rotation Toit 2 4 F Reduction Rectangle -toit 2 4 G Rotation Rectangle -toit 2 4 D Symetrie Droite -toit 2 4 E Rotation Droite -toit 2	Bleu
7	3 Z Mur 2 4 Moyen Droite -toit 2 4 Rouge Rectangle - toit 2 4 Mauve Droite - toit 2	3 Y Translation Mur 2 4 F Reduction Rectangle -toit 2 4 G Rotation Rectangle -toit 2 4 D Symetrie Droite -toit 2 4 E Rotation Droite -toit 2 4 X Symetrie Droite -toit 3	bleu

EXEMPLE ! tables 2eme partie

Etape	Table locaux couleur	Table locaux trait	Table locaux transferables	Ecran
8	Idem	Idem	Idem	bleu
9	3 Z Mur 2 4 Rouge Rectangle - toit 2		3 Y Translation Mur 2 4 F Reduction Rectangle -toit 2 4 G Rotation Rectangle -toit 2	Bleu
10	Idem	Idem	Idem	bleu
11	Idem	Idem	Idem	bleu
12	Idem	Idem	Idem	bleu
13	3 Z Mur 2 4 Rouge Rectangle - toit 2		3 Y Translation Mur 2 4 F Reduction Rectangle -toit 2 4 G Rotation Rectangle -toit 2	Bleu
14	3 Z Mur 2		3 Y Translation Mur 2	Bleu
15			4 M Symetrie Rectangle 3 4 N Dilatation Droite 3	Bleu
16			4 M Symetrie Rectangle 3	bleu
17	Idem	Idem	Idem	bleu
18	Idem	Idem	Idem	bleu
19	Idem	Idem	Idem	bleu

EXEMPLE : tables 2eme partie

Etape	Table locaux couleur	Table locaux trait	Table locaux transferables	Ecran
20				bleu
21				bleu
22				bleu
23	3 Jaune Mur 2	3 Moyen Mur 2	3 H Dilatation Mur 2 3 I Reduction Mur 2	bleu
24	3 Jaune Mur 2	3 Moyen Mur 2	3 H Dilatation Mur 2 3 I Reduction Mur 2 4 X Symetrie Droite 3	bleu
25	Idem	Idem	Idem	bleu
26	3 Jaune Mur 2	3 Moyen Mur 2	3 H Dilatation Mur 2 3 I Reduction Mur 2	bleu
27	Idem	Idem	Idem	bleu
28	Idem	Idem	Idem	bleu
29	Idem	Idem	Idem	bleu
30	Idem	Idem	Idem	bleu
31	Idem	Idem	Idem	bleu
32			4 M Symetrie Rectangle 3 4 N Dilatation Droite 3	bleu
33			4 M Symetrie Rectangle 3	bleu

EXEMPLE : tables 2eme partie

Etape	Table locaux couleur	Table locaux trait	Table locaux transferables	Ecran
34	Idem	Idem	Idem	bleu
35	Idem	Idem	Idem	bleu
36	Idem	Idem	Idem	bleu
37				bleu
38				bleu